0-315-01119-X

National Library of Canada / Bibliothèque nationale du Canada

Canadian Theses Division / Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

48977

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

SREEKAANTH S. ISLOOR

Date of Birth — Date de naissance	Country of Birth — Lieu de naissance
MARCH 27, 1951	INDIA

Permanent Address — Résidence fixe

C/O SREEKAANTH S. ISLOOR
BELL NORTHERN RESEARCH
CAD DATABASE DEVELOPMENT DEPT.
P.O. BOX 3511, STATION C
OTTAWA, CANADA, K1Y 4H7

INDIA

Title of Thesis — Titre de la thèse

CONSISTENCY ASPECTS OF DISTRIBUTED DATABASES

University — Université

UNIVERSITY OF ALBERTA

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

DOCTOR OF PHILOSOPHY

Year this degree conferred — Année d'obtention de ce grade	Name of Supervisor — Nom du directeur de thèse
1979	T. A. MARSLAND

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date	Signature
August 23, 1979	Sreekaanth S Isloor

NL-91 (4/77)

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

Ottawa, Canada
K1A 0N4

NL-339 (Rev. 8/80)

THE UNIVERSITY OF ALBERTA

CONSISTENCY ASPECTS OF DISTRIBUTED DATABASES

by

SREEKAANTH S. ISLOOR

(C)

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA, CANADA

FALL, 1979

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "Consistency Aspects of Distributed Databases" submitted by SREEKAANTH S. ISLOOR in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

T.A. Marsland
.....
Supervisor: T.A. Marsland

Daniel H. Bent
.....
D. Bent

D. Fenna
.....
D. Fenna

.....
L.W. Jackson

Dennis K. Hsiao
.....
External Examiner: D.K. Hsiao

DATE.....23. Aug. 1979.....

to my parents,

Srinivas Rao and Shalini Isloor

ABSTRACT

The consistency of data is threatened by (i) inadequate concurrency controls, (ii) system crashes, (iii) security breaches, and (iv) erroneous software. With the growing use of terminal-oriented computer systems and with the trend towards the distribution of system functions over a computer network, preservation of consistency of the database is one of the most critical problems faced by the designers of the distributed information networks. In this thesis, a detailed understanding of the problems caused by inadequate concurrency controls and system crashes in a network environment is considered.

A new approach to deadlock detection is proposed. The concept of "on-line" deadlock detection in distributed information networks is introduced. It is defined to be the process of recognizing deadlock occurrence as soon as it happens, at the installation which makes the resource allocation decision, without the necessity of further communication for every request made or granted. An on-line detection algorithm is suggested and developed. All of the earlier algorithms restrict a process to having at most one outstanding request. In our approach, such a restriction is removed in view of the fact that in real-world applications more than one outstanding request is a certainty. This leads to a situation in which an allocation decision on a

data resource (with multiple waiting-access requests) released by a completing process would lead to a deadlock. For this case, the results and the approach suggested are new and original. An elegant solution which combines the principles of detection and avoidance, first of its kind as a mixed method in database systems, is shown to detect and avoid a potential deadlock.

Another aspect of the problem considered is the reliable operation of database systems, partitioned and/or replicated over a network of computers. The design of a method which maintains database consistency during system update and recovery is guided by the goals of simplicity, tolerable overhead, partial operability, and avoidance of global rollback. In this new approach, retrieval and update transactions are subclassified and recovery protocols defined which take advantage of the known properties of each transaction class. An optimal policy for checkpointing in a particular recovery protocol is derived using a new simple model. The policy determines the checkpoint dynamically as the transactions are processed, and is different from earlier fixed interval methods. The feasibility of its implementation makes the scheme new and practical. The cascading effect of a global rollback is modeled by using the progress of processes represented by a set of interaction tuples and recovery points ordered in the time domain. A backup algorithm based on this model is

developed. Recovery aspects for a wide set of system failures are considered and several partial solutions are outlined.

ACKNOWLEDGEMENTS

Professor T.A. Marsland started out initially as my instructor in two courses, and became my supervisor, and friend. I have benefitted as immensely from his constructive criticisms on parts of my thesis as I have from his invaluable advice on several other qualities in life. The best wish I could offer to any graduate student would be to have a supervisor like Dr. Marsland. THANK YOU TONY.

I thank Professor David K. Hsiao of the Ohio State University, and Professors Dale Bent, Donald Fenna and Wayne Jackson for their useful comments on this thesis.

I am grateful to Mr. Christopher Gray for his careful reading of the initial drafts of certain chapters. Several people have contributed in one way or the other in creating an excellent work environment for me in the past four years. The three individuals who deserve my thanks are, Dr. John Tartar, Dr. Keith Smillie, and Mr. Robert Willoner.

TABLE OF CONTENTS

CHAPTER		PAGE
1:	INTRODUCTION	1
1.1	Distributed Systems	1
1.2	Distributed Databases	4
1.3	Centralized *versus* Distributed Approach	6
1.4	Maintenance of Semantic Correctness of Data	8
1.5	Overview and Outline of the Thesis	14
2:	DEADLOCKS IN OPERATING, DATABASE, AND DISTRIBUTED SYSTEMS	16
2.1	The Deadlock Problem	16
2.2	Examples of Deadlocks	18
2.3	Approaches to Handling Deadlocks	25
2.4	Models of Deadlock	32
2.5	Comparison and Contrast of Deadlock Problem in the Three Fields	38
2.6	Survey of Deadlock Handling Schemes	41
2.6.1	Operating Systems	41
2.6.2	Database Systems	45
2.6.3	Distributed Databases	49
2.7	Combined or Mixed Approach to Deadlock Handling	52
2.8	Deadlock Problem *vis-a-vis* Game Playing	55
2.9	On Probabilistic Model of Deadlock	58
2.10	Current Implementations of Deadlock Handling Schemes	59

3: 'ON-LINE' DEADLOCK DETECTION IN DISTRIBUTED DATABASES 63

3.1 Graph-Theoretic Model and Concepts 63

3.2 A Running Example 65

3.3 Necessary and Sufficient Conditions for Deadlocks 67

3.3.1 Characteristics of Waiting Processes 71

3.4 Deadlock Detection in Distributed DBMS 73

3.5 'On-Line' Detection 75

3.6 Resolution of Process-Resource Interactions ... 78

3.7 'On-Line' Deadlock Detection Algorithms 84

3.8 Discussion 92

3.8.1 Communication Requirements 92

3.8.2 Responsibilities of a Data Base Administrator 94

3.9 Highlights of the Proposal 96

4: SYSTEM RECOVERY IN DISTRIBUTED DATABASES 98

4.1 Introduction 98

4.2 The Problem, Environment, and Basic Strategy .. 99

4.3 Transaction Classification 104

4.3.1 Retrieval Transactions 104

4.3.2 Update Transactions 105

4.4 System Recovery Protocols 108

4.5 Optimal Policy for Checkpointing 112

4.5.1 Audit Trail Maintenance Policy 116

4.6 Domino Effect (Global Rollback) 117

4.7 Transaction Processing Model 120

4.8 The Backup Algorithm 121

4.9 Recovery from Different Failures 123

4.10 Discussion 126

5: CONCLUDING REMARKS 128

5.1 Summary of the Results Obtained 128

5.1.1 Deadlocks 128

5.1.2 System Recovery 129

5.2 Significance and Motivation 130

5.3 Directions for Research 133

BIBLIOGRAPHY 137

APPENDIX A: AN EXAMPLE OF "CYCLIC RESTART" 144

APPENDIX B: AN EXAMPLE FOR MULTIPLE OUTSTANDING REQUESTS 146

LIST OF TABLES

Table	Description	Page
2.1	Summary of Detection, Prevention, and Avoidance Approaches	29
2.2	Summary of Deadlock Handling Techniques in Distributed Databases	30

LIST OF FIGURES

Figure	Description	Page
1.1	A Distributed Information Network	5
2.1	Traffic Deadlock at an intersection marked with 4-way stop signs	20
2.2	Deadlock in Consistent Database Systems	23
2.3	State Graph for Example 2.1	36
2.4	General Resource Graph for the traffic deadlock of Figure 2.1	37
2.5	Distributed Database Deadlock: Example of Figure 2.2 in distributed environment ..	40
3.1	System Graph for a Running Example	66
4.1	Domino Effect	120

CHAPTER 1

INTRODUCTION

1.1 Distributed Systems

The steady "affair" of the past decade between computers and communication technology has matured into a "marriage" bringing with it the creation of large computer communication complexes. Consequently, we have witnessed in recent years the rapid evolution of computer communication networks from research efforts to operational utilities. There is hardly any doubt that networks of computers will play an increasingly important role in providing enhanced computing services to users in universities, business firms, and government agencies. For instance, the ARPANET [McQuillan and Walden, 1977], currently supports communication among more than one hundred computer systems. The network is under continual development and is used by thousands of users daily. Computer networks have the capability to bring computing power to the people who need it, and provide access to a wider variety of resources dispersed among several computers linked by a communications facility to provide the basis for a "distributed" computing service. Potentially, this technology will revolutionize

the way data processing is done. Bright futures have been predicted for commercial networks such as TELENET [Hovey, 1974] in the United States and DATAPAC [Clipsham et al., 1976] in Canada.

A system is said to be distributed if hardware or processing logic, data, the processing actions or the operating system components are dispersed on multiple computers which are logically and physically interconnected. In such a system, data may be replicated at several sites or on separate storage devices; the processing logic cooperates and interacts through a communication network under decentralized system-wide control. A taxonomy of distributed processing systems design, applications, models, effects, experiences and techniques has been provided elsewhere [Chang, 1976; Le Lann, 1977; Maryanski and Kreimer, 1978; Eckhouse et al., 1978; van Dam and Stankovic, 1978; Marsland and Sutphen, 1978]. The essential properties and characteristics of a distributed system are summarized below.

* A wide variety of general-purpose resources, both physical and logical in nature, is available. These resources can be dynamically assigned to specific transactions. However, some special-purpose dedicated resources may not be reassignable.
* The notion of autonomous operation of the process is supported by the distribution of physical and logical resources of the system, interacting with processes

through a communication complex. The transfer of messages follows the principle of a two-party protocol. In this protocol, the cooperation of the two parties is essential to complete a transfer successfully.

* The various processors may have non-homogeneous operating systems. A high-level operating system, in the form of a collection of well-defined protocols and software, governs the integrated functioning of the network. However, the autonomous operation of each computer requires the absence of a strong hierarchy between the network operating system and the local operating systems.

* The interface between the user and the distributed system provides transparency from the system organization. Effectively, the user can handle resources as if he were communicating with a single, centralized system. A high level interface provides data independence in systems, and hides system status from the user. However, provision can be made for a knowledgeable user to request services by the designation of the server.

* Cooperative autonomy rather than independent behaviour is the manner in which the system functions. All the processors follow the general guidelines outlined in the network operating system to facilitate this cooperation. Autonomous functioning at both logical and physical levels is offered by this essential component of the system.

1.2 Distributed Databases

An estimated 20% of the United States G.N.P. is devoted to the collection, processing and dissemination of information[1], leading to data management occupying an important segment of the United States economy. Surprisingly, only a small portion of this information is computerized. With such tremendous potential requirements of automated data management, along with the increases in database size, complexity and diversity of use, and users' strong preference for interactive computer systems, the necessity for additional computing resources grows rapidly. Replacement by higher performance components is an expensive way of growing. Distributing several system functions and data over a network of computers has been projected as an economic panacea to the expansion problem which will provide improved performance, as well as enhanced accessibility of data [Booth, 1972, 1976; Comba, 1975; Enslow, 1978]. The evolution of modern computer network technology and the rise of common carrier packet switched networks have provided motivation to the development of distributed information networks. The increased reliability that can be achieved with distributed databases and the availability of inexpensive mini- and micro-computers due to falling hardware costs, have contributed to the widespread interest in such systems. Distributed databases will clearly meet

---

1: Frost and Sullivan, Inc., Markets for Data Base Services, New York, July, 1973, pp. 11.

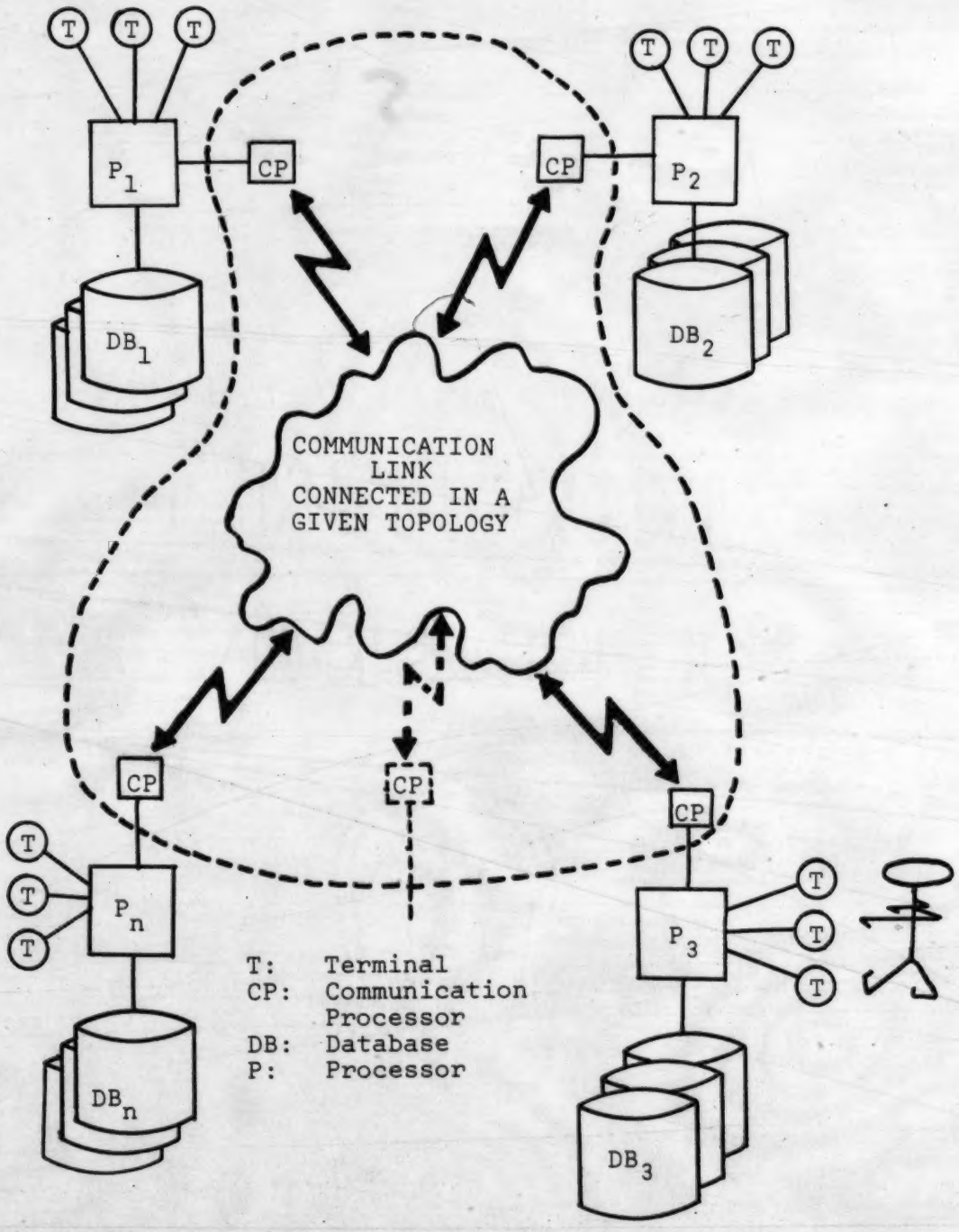


Figure 1.1: A Distributed Information Network

the future challenges of effective information management. Complete reviews of distributed database management have appeared in several articles [Deppe and Fry, 1976; Lowenthal, 1977; Rothnie and Goodman, 1977; Peebles and Manning, 1978; Maryanski, 1978; CODASYL, 1978; Davenport, 1978; Adiba et al., 1978; Rothnie et al., 1979].

Typically, a distributed database management system consists of two or more computers interconnected by a communication network. Each computer has a database attached to its auxiliary storage. An example of a distributed database is schematically depicted in Figure 1.1. Each processor is either a general-purpose system, or a backend [Canaday et al., 1974], or a database machine [Baum and Hsiao, 1976; Banerjee, Baum and Hsiao, 1978; Hsiao, 1979]. Front-end communication processors are interconnected by communication links, such as coaxial cables, wire pairs, or microwave channels. These front-ends provide the interface through which host computers are connected to the communication facility, and also cooperate to support communication between hosts. Each host consists of a database management system which supports one or more interactive users.

1.3 Centralized versus Distributed Approach

Terminal-based computer communications systems, in the past, have generally focussed around a single, large computer installation. Although a fair argument can still

be made for servicing remote users with a centralized system, the major deficiencies of such a system have contributed to widespread belief in the distributed approach. With a centralized system data communication costs are significant. The steady cost of transmitting data in contrast to the drastic fall in hardware costs further motivates decentralization of data management. Despite economic benefits of centralized systems in areas such as operations, managerial personnel have become aware of general undesirable side effects of such a system. The increasing demand of a growing community of remote users for interactive systems and the consequent need to deal with more concurrent events leads to enhanced complexity in servicing with a centralized system. Moreover, the failure of the central node brings an organization to a standstill. The requirement to endorse and enforce standardized, centralized data processing project development often works contrary to the management philosophy and needs of hierarchically decentralized organizations.

Besides overcoming the above perceived shortcomings, distributed databases offer several major advantages. These are generally regarded as including:

(a) Reliability: With data redundantly stored on multiple computers, the system is not susceptible to total failure when a single computer component breaks down.

(b) Responsiveness: The close proximity of data enhances accessibility of resources and improves system

performance. A higher throughput is obtained because of parallel processing.

(c) Shareability: The ability to share data among several geographically separated installations and to gain access to specialized resources that would otherwise not be available or would be available at an unacceptable cost is enhanced.

(d) Expansion: The system lends itself to incremental upward scaling. The system can be developed on an incremental basis with only as much computing power being installed as is required at that time. Distributed systems have the capability to absorb new technology as it is invented and will not necessitate that in-place systems be returned to the vendor or scrapped.

(e) Human factors: Individual groups can physically possess part of the corporate database (which holds their part of the data) giving themselves the responsibility and the satisfaction of updating their own database. Another favourable human factor is the reduction of the vulnerability of the database to strike action or acts of terrorism.

1.4 Maintenance of Semantic Correctness of Data

A database is regarded as a model of some limited universe rather than just a collection of values. At every instant of time, some configuration of that application world is reflected by the contents of the database. Whether

any given configuration of the database is reasonable or not is specified by a set of rules. Ensuring the validity of the set of constraints means to guarantee the integrity of data, or rather more precisely, data accuracy, consistency and timeliness.

When the database contents comply with constraints derived from the knowledge about the meaning of the data, it is said that the semantic correctness of the data is preserved. Semantic correctness can be enforced by allowing on the database only a set of precisely specified meaningful operations; by adopting a set of programming and interaction conventions; by dynamically checking the results of updates; or by proving for each update process that the integrity constraints are satisfied.

Loss of correctness (inconsistency) can be seen in several forms. In its simplest form, an individual value of a particular field can be inappropriate. For instance, a salary which is non-numeric or a birth year which places the person's age at 250 years. Secondly, an inconsistency between different fields of the same record, such as an individual's salary not reflecting his professional status. Thirdly, an inconsistency may occur between the field(s) of one record and field(s) of related record(s). For example, in an employee database, a rule that managers draw more than their employees may be violated. Fourthly, certain global patterns may be out of order in some set of records. Such an inconsistency is not due to the individual records, but

due to a collection of them. The global patterns can typically be aggregate functions, for instance, the average salary of all employees is less than $20000, or every department has exactly one manager. Normally these violations are not due to a faulty value, but due to noncompliance with expected patterns. Lastly, blank fields, obsolete values or records that cannot be found are incorporated in the notion of missing data.

Users of shared databases presume that the correctness of the information upon which they work is preserved. Preservation of such consistency is one of the most critical problems faced by the designers of database systems. The semantic accuracy of the data is threatened by (i) inadequate concurrency controls; (ii) system crashes; (iii) security breaches; and (iv) erroneous software.

Concurrency Controls:

Shared access is allowed to the database to maximize concurrent use of system resources. Concurrency control is a system mechanism that is concerned with deciding what actions should be taken in response to requests by individual processes (transactions) to update the database. The concurrency control should be capable of effectively handling conflicting update requests, deadlocks or similar occurrences, and maintain consistency of the database. In a distributed database the update mechanism must guarantee that updates to database copies preserve the mutual

consistency of multiple copies of the replicated data as well as maintain the internal consistency of each database copy. Mutual consistency requires that all copies of the replicated data be identical, in the sense that they must converge to the same final state if all user activity were to cease. Internal consistency requires the maintenance of semantic accuracy of data, just as in a non-redundant database.

Consider the data entities x, y and z duplicated at installations N_1 and N_2. Let the initial values of all these entities at both installations be 1. Further, let us assume that internal consistency constraint is that "the sum of the values of x, y, and z is 3". Consider the two updates U_1 and U_2.

U_1: x <-- 0, y <-- 2;
U_2: y <-- 0, z <-- 2;

If U_1 and U_2 are both applied, the internal consistency of the database will be destroyed, regardless of their order of application. Proper operation dictates the rejection of one of the updates. The mutual consistency of the data would be destroyed if updates U_1 and U_2 are accepted at N_1 and N_2 respectively, even though the internal consistency of each copy is preserved. The maintenance of consistency requires some sort of locking scheme, which in turn leads to deadlocks.

System Crashes:

The problem of reliable operation of a distributed database system in the presence of failures, boils down to maintaining database consistency during update transactions in the presence of either system failure or communication breakdown. The wide spectrum of system malfunctions are of the form: (a) a storage failure (head crash); (b) a system crash (software failure); (c) a lost message; (d) a duplicated message; (e) a lost process (due to system crash and subsequent recovery); (f) network partitioning; and (g) operation with missing nodes.

Suppose for instance, a database transaction updates three records, each stored at a different installation. The updates are not in effect until all have been completed and acknowledged. After receiving the acknowledgement from individual installations, the source of the transaction communicates the message to effect the updates to the three installations. Typically, the problem of maintaining consistency arises when one of the installations crashes before receiving the message.

Security Breaches:

Potential security violations can be categorized into the following three classes [Saltzer and Schroeder, 1975]:

(a) Unauthorized information release: refers to unauthorized access by a person to take advantage of data stored in the computer. An intruder can infer correctly by

observing the patterns of input and output by what can be termed "data-traffic analysis". Tapping of network communications may result in unauthorized exposure of sensitive information, alteration of message text, misrouting or misdelivery of messages, or spoofing of a network resource.

(b) Unauthorized information modification: relates to the ability of an unauthorized person to change or delete the information stored in the computer. Network penetrators may be able to use counterfeit network resource.

(c) Unauthorized denial of use: an intruder can deny the authorized user the privilege to access or update due information by sabotaging the system. The forms of sabotage may be by causing the system to 'crash', by disrupting the transaction manager, or by directly firing a bullet into the computer.

Unauthorized release, modification, or denial of use can potentially result in the loss of information itself, or loss of semantic accuracy of the data.

Erroneous Software:

It is essential to verify and to test that the software produced is indeed the software intended. Such verification can be achieved by proof of correctness and penetration tests. Faulty software may compromise the consistency of the data on which it acts. The term 'fault tolerant system' is normally used to denote the system software which will

continue to yield correct results in the presence of hardware and software fault conditions. The discussion of this area is limited since the topic is beyond the scope of this thesis.

1.5 Overview and Outline of the Thesis

In the early stage of the thesis, a complete overview of the deadlock problem is provided. There is a detailed discussion of the essential characteristics, relationships and models of the deadlock problem in operating systems, database systems, and distributed databases. The value of combined solutions which incorporate detection, avoidance and prevention principles is examined. Comments on the game theory approach to deadlock, and probabilistic models of deadlock are presented.

Holt's[1972] graph theoretic model is used and extended for the distributed database case and a new algorithm to detect deadlocks using this model is proposed. The concept of 'on-line' detection of deadlocks in distributed databases is introduced and defined. An effective algorithm to detect deadlocks on-line, which allows the data resource allocation decision to take place without further reference to other computers is proposed. A realistic approach is taken, allowing processes to have more than one outstanding resource request simultaneously. This approach combines the principles of detection and avoidance as well.

For recovery from system crashes, the knowledge of the nature of all the processes that access the data is effectively utilized to design system recovery protocols. These protocols reflect the importance of the availability of the on-line system at all times, and advocate that the recovery overhead should be directly dependent on the value or sensitivity of data accessed. An optimal policy for checkpointing dynamically using a new simple model is derived. The "domino effect" (cascading effect of global rollback) is modeled, and a rollback algorithm is developed.

Finally, the overall significance of all the results and an overview of the motivation for the research is outlined. Outstanding problems and areas that require further detailed solutions are indicated.

CHAPTER 2

DEADLOCKS IN OPERATING, DATABASE, AND DISTRIBUTED SYSTEMS

2.1 The Deadlock Problem

Modern multiprogramming systems are designed to support a high degree of parallelism by ensuring that as many system components as possible are operating concurrently. The increasing trend by commercial firms for on-line operations, especially those involving integrated databases, and the consequent need by active users for a responsive system, have placed heavy demands on database-oriented operating systems. Compounding these difficulties is the arrival of distributed processing as a solution to incremental system growth. One characteristic of such contemporary systems is their high degree of resource and data sharing. Concurrency must be regulated by some facility which controls access to sharable resources. Computerized information systems typically use locks [Gray, 1974; Gray et al., 1975, 1976] for this purpose. A simple lock protocol associates a lock with each data object. A process locks the object it uses and holds it until the successful completion of the transaction. The lock has the effect of notifying others that the object is busy. Deadlocks arise when members of a

process group are requesting access to resources already held exclusively by other processes within the group. When no such member of the group is willing to relinquish control over its resources until after it has completed its resource acquisition, deadlock is inevitable, and can only be broken by the involvement of some external agency.

A set of processes becomes deadlocked as a result of the presence of certain conditions, which may be informally summarized as the exclusive access and the circular wait conditions. The simplest illustration of these involves only two processes, each holding, for exclusive access, a different resource and each requesting access to the resource held by the other. The result is a circular wait which cannot be broken until one of the processes releases the resource it holds, or cancels the request it made.

In a more general case, a circular wait state involving a set of processes is said to exist when these processes are linked in a circular chain in such a way that every process holds at least one resource and is waiting for at least one more resource held by the next process in the chain. A circular wait condition may arise once the following necessary conditions hold:

* each process requests exclusive control of one or more resources (like printers, tape drives, data records for updating, etc.);
* the processes hold resources allocated to them, while seeking additional ones (data resources should be held

until process completion for consistency reasons [Eswaran et al., 1976]); and

* the preemption of resources from processes cannot be done without aborting the processes (preemption is the reclaiming of a resource by the system and requires the support of a rollback and recovery mechanism, especially when data resources are involved).

2.2 Examples of Deadlocks

The deadlock problem occurs in many different contexts. Analogies can be made to real life situations, provided one interprets the processes and resources involved appropriately. For instance, one often hears about the "deadlocked" peace talks between two countries which were at war. In that context, the peace-negotiating parties of the two countries are the processes, whereas the occupied territories are possibly the resources over which exclusive control is sought. The peace talks could be deadlocked if both parties refuse to give up any occupied lands until after the return of some land which their adversary holds. Several other examples of deadlocks arise in day-to-day operations in the real world.

Another common example is that of the traffic deadlock. At uncontrolled intersections, or at intersections marked with 4-way stop signs, traffic regulations require that, when two or more vehicles approach or enter the intersection on adjacent roads at approximately the same time, the driver

of the vehicle to the left shall yield the right of way to the vehicle on the right.

Consider the situation depicted in Figure 2.1, where the four cars A, B, C, and D have arrived at an intersection marked with 4-way stop signs at approximately the same time. According to the traffic regulation A should yield to B, B to C, C to D, and D to A. As far as A is concerned, part of the intersection belongs to B for exclusive use, so A must wait until B passes, and so on. It is evident that a circular wait exists as an essential component of this traffic deadlock. Note that if all four vehicles move forward, occupying as much of the intersection as possible, all traffic comes to a standstill. Many major cities avoid this difficulty by cross hatching important intersections requiring that no vehicle enter that area unless its exit route is clear. In this illustration, the cars A, B, C, D are the processes, while the space in the intersection is a resource to which each needs exclusive access. The cross hatching technique is a means of ensuring that no process will acquire a resource (i.e. occupy the intersection) which it cannot subsequently relinquish (i.e. leave by a clear exit) without loss of control or function.

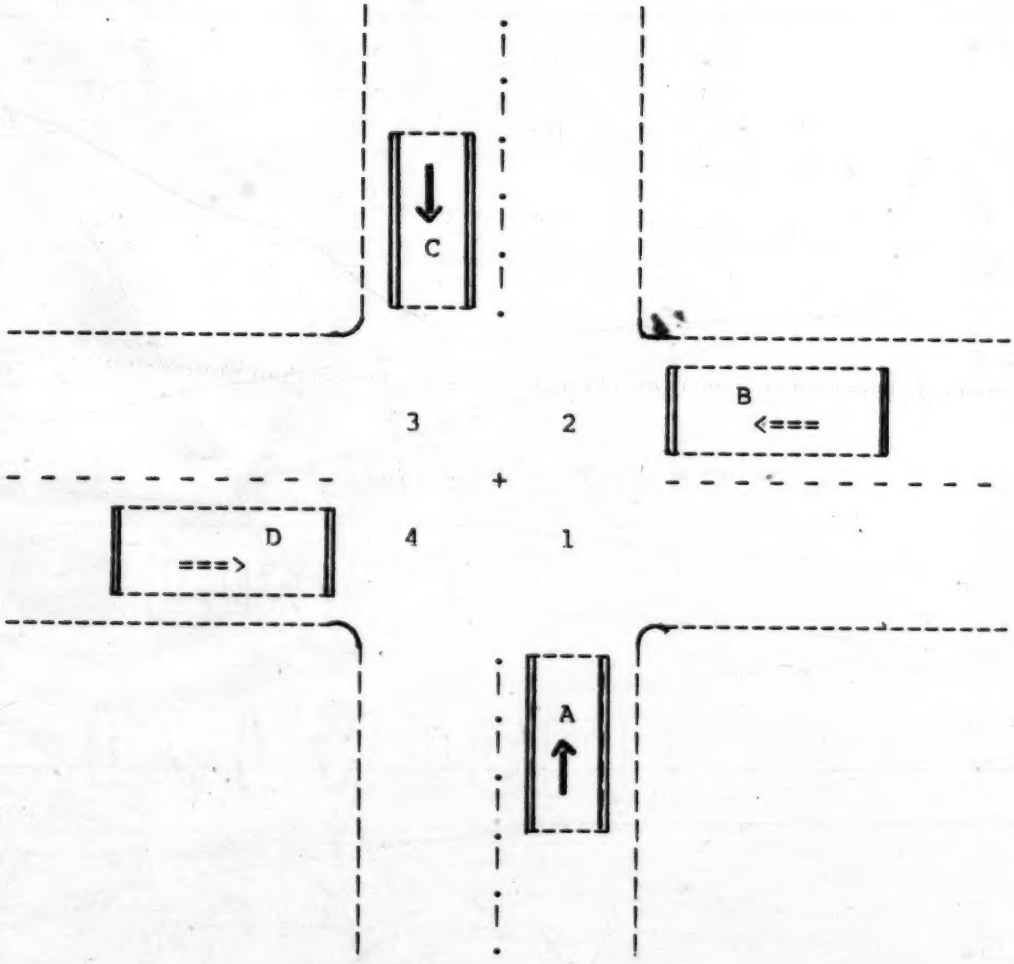


Figure 2.1: Traffic Deadlock at an intersection marked with 4-way stop signs.

If the four quarters of the intersection are marked 1, 2, 3, and 4, the deadlock situation is different depending on whether the cars turn right, turn left, or go straight. Assuming the exit route for each car outside of the intersection is clear, the apparent deadlock can be handled easily in the case when all the cars wish to make a right turn. Each car requires access to one quarter of the intersection which no others demand. For instance, in

Figure 2.1, cars A, B, C, and D need access to quarters 1, 2, 3, and 4 in the intersection respectively. Actually, in this case, there is no resource contention.

The state of affairs is different and difficult when all the cars either go straight or turn left or a combination of the two. Car A requires quarters 1 and 2 of the intersection to go straight. Similarly car B needs quarters 2 and 3, and so on. In these circumstances, the deadlock arises due to each car requiring exclusive access to the quarter of the intersection to which the car on its right side has legal right. Each car requires two resources (considering each quarter of the intersection as a different resource), one of which is held by the car on its right. The traffic deadlock condition worsens if all the cars desire to turn left. For instance, car C requires the use of quarters 3, 4, and 1 of the intersection. Quarters 4 and 1 are legally held by the cars D and A respectively. Similarly car D needs the quarters 4, 1, and 2, of which quarters 1 and 2 are held by cars A and B. In this traffic case, the deadlock arises because of each car being blocked by two other cars, one on its right side and the other straight ahead. Each resource (every quarter of the intersection) has two waiting processes (cars) besides the one which legally holds it for access.

The traffic situation can be resolved through the interference of an external agency (a policeman) allocating space to one of the vehicles. As a practical matter the

traffic deadlock case is usually resolved by one driver aggressively entering the intersection. However, an alternative solution might be to require vehicles to back up a random amount and approach the cross roads again, thereby, breaking a simultaneous arrival. A similar technique is used in ETHERNET [Metcalfe and Boggs, 1976] to resolve access conflicts on their communication medium, and is usual on contention-mode telecommunication circuits, except that "random" means varied but fixed.

Of greater interest, perhaps, are the deadlocks that may occur in computer operating and database systems. The term "system resource" in computers broadly refers to storage media (like primary memory, tapes, disks, drums, etc.), system components (such as tape drives, disk drives, I/O channels, CPU, readers and printers, etc.), and information (for example communication messages, data records, files, directories, programs, system routines, etc.). Consider a small multiprogramming system with a single card reader and a printer, in which two user jobs share use of the printer and the card reader by means of request and release operations, as given in standard operating systems texts. Due to independent scheduling of the jobs the request and release operations can be interspersed in several different orders. Some of these sequences lead to a "deadly embrace" due to jobs holding respectively the printer or the card reader, and at the same time requesting the unit held by the other.

In the database case, consider two concurrently executing processes P and Q, which modify entities M and N. For example:

P P_1: M <== M + 100
P P_2: N <== N + 100

Q Q_1: N <== 2 * N
Q Q_2: M <== 2 * M

PROCESS P

STEP	ACTION
p_0	REQUEST ENTITY M
p_1	LOCK ENTITY M
p_2	READ ENTITY M
p_3	WRITE ENTITY M
p_4	REQUEST ENTITY N
p_5	LOCK ENTITY N
p_6	READ ENTITY N
p_7	WRITE ENTITY N
p_8	UNLOCK ENTITY M
p_9	UNLOCK ENTITY N

PROCESS Q

STEP	ACTION
q_0	REQUEST ENTITY N
q_1	LOCK ENTITY N
q_2	READ ENTITY N
q_3	WRITE ENTITY N
q_4	REQUEST ENTITY M
q_5	LOCK ENTITY M
q_6	READ ENTITY M
q_7	WRITE ENTITY M
q_8	UNLOCK ENTITY N
q_9	UNLOCK ENTITY M

Sequence of steps that leads to deadlock:
$p_0q_0p_1q_1p_2p_3q_2q_3p_4q_4$

Figure 2.2: Deadlock in consistent database systems.

Let us assume that the database correctness (consistency) assertion on the entities is that M = N, and that the initial values of M and N are the same. Interleaving the actions of processes P and Q in an arbitrary fashion, such as P_1-Q_1-P_2-Q_2, leads to the loss of database correctness.

Although it is possible to construct the processes so that database correctness is maintained, concurrent operation can still lead to deadlock. In order to retain the "strong consistency" result of Eswaran et al. [1976], which requires that the processes be "well-formed" and "two-phase", the processes P and Q are divided into disjoint locking and unlocking phases. A well-formed process is one which locks an entity before acting on it further, and subsequently unlocks such entities. A process is thus required to be divided into growing and shrinking phases. The first unlock action marks the beginning of the shrinking phase, after which a process cannot issue a lock request on any entity in the database until the release of all entities held by the process. In this context, it is essential to note that the process P (or Q) cannot unlock entity M (or N) before locking entity N (or M), if it is to maintain database correctness. The actions of processes P and Q, and a sequence of steps that would lead to a deadlock under concurrent operation, are shown in Figure 2.2. Besides this illustration, the example is used again in Section 2.5 to demonstrate the deadlock problem which could arise in a distributed database environment. Several other aspects of concurrent operation such as transaction, lock, log, and recovery management have been dealt with thoroughly elsewhere [Gray, 1978], and are beyond the scope of this thesis.

2.3 Approaches to Handling Deadlocks

The basic strategies for handling deadlocks are detection, prevention and avoidance (by heuristic means).

Detection Techniques:

When deadlock detection schemes are used the requested resources are granted where possible. These techniques periodically invoke an algorithm which examines the current resource allocations and outstanding requests, in order to identify any processes and resources involved in a deadlock. If a deadlock is discovered, the system must recover as gracefully as possible by preempting resources from relevant processes until the deadlock is broken.

The overhead involved in detection comprises of the run-time cost of the algorithm and the potential losses inherent in preempting resources. In detection schemes no action is prompted until the actual occurrence of a deadlock. Thus a resource may be held idle by a blocked process for a long period of time. If the resource held is a tape drive, for which preemption involves possibly unacceptable overhead, it is difficult to use detection principles effectively.

Nevertheless, detection techniques have some advantages, since the scheme is invoked intermittently. In contrast, avoidance or prevention mechanisms have to ensure that deadlocks never occur for any request made, resulting

in undue process waits and run-time overhead. On-line handling of deadlocks is facilitated by detection principles as developed in Chapter 3. Detection is used in conjunction with a resumption mechanism, such as a task swapping facility. Resource preemptions are minimal since only the essential ones occur.

In the context of database systems or distributed databases, detection methods rely on the management system to abort, rollback, and restart at least one database process to break the deadlock. Here, the problem of rollback and recovery assumes great importance from the viewpoint of maintaining consistency of the database.

Prevention Mechanisms:

Prevention is the process of systematically structuring the requests of processes in a fashion such that deadlocks will never occur, by putting constraints on the system's users. Most proposals for prevention require that each process specify, a priori, all the resources needed before processing begins. A prevention mechanism differs from an avoidance scheme in that performing run-time testing of potential allocations is not necessary. The deadlock can be prevented in several ways, including requesting all resources at once, preempting of resources held, and resource ordering.

The simplest method of preventing deadlocks is to outlaw concurrency, which is an administrative solution to

the problem. However, it is not consistent with the present day philosophy of system design and leads to very poor usage of resources. Another method requires that all resources be acquired before processing starts. Such a scheme is inefficient since resources held may be idle for prolonged periods. This method works well for processes which perform a single burst of activity, such as input/output drivers. For processes with fluctuating requirements the method may be impractical. In a database environment, it may be impossible for a very highly data driven process to specify and acquire all needed resources before beginning processing. In any case, the scheme discriminates heavily against data driven processes.

Certain other prevention methods require a blocked process to release a resource held, when requested by an active process. In such schemes a process goes from an active to a blocked state when its request for some resource cannot be satisfied immediately. A typical example is the usage of main memory, where a process is completely swapped out by preempting the memory it holds, whenever more than the currently available memory is requested. The process is swapped back only when the entire larger quantity of memory is available. In this scheme preemption of resources is done more often than necessary. Under certain circumstances in database systems, this scheme is subject to what is called "cyclic restart" [Stearns et al., 1976] in which two or more processes continually block, abort, and restart each

other, as shown with a detailed example in Appendix A.

A more refined form of prevention is by resource ordering [Habermann, 1969]. All resources are uniquely ordered, and a request for a specific resource is met if and only if all resources lower in the ordering that are needed in the future are also allocated. Blocking of processes in a circular wait is ruled out by the ordering rule. The feasibility of enforcing resource ordering by compile-time checks is a major advantage of the scheme. Run-time computation is not needed as the deadlock problem is solved completely in the system design. Besides eliminating the overhead associated with an avoidance algorithm the scheduling problem is simplified. Sequences of requests by a process which can be allowed are restricted by the scheme. This method relies heavily on educating system users regarding the ordering rule. A process may request and hold a resource rather early in the processing stage, in which case an incremental request for the same resource at a later stage is disallowed by the ordering rule. This leads to preempting all the resources held by the process. In a database system environment with processes of fluctuating needs, it is difficult and almost impossible to order the data resources (records, entities, or fields).

Avoidance Schemes:

In avoidance schemes, a request by a process for a resource is granted if and only if it is certain that this

TABLE 2.1: SUMMARY OF DETECTION, PREVENTION, AND AVOIDANCE APPROACHES

Principle	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
DETECTION	Very liberal; requested resources are granted, where possible.	Invoke periodically to test for deadlock.	*Process initiation never delayed. *Facilitates "on-line" handling.	*Inherent pre-emption losses.
PREVENTION	Conservative; under commits resources.	Requesting all resources at once	*Works well for processes that perform a single burst of activity. *No Preemption necessary.	*Inefficient. *Process initiation delayed.
		Preemption.	*Convenient when applied to resources whose state can be saved and restored easy.	*Preemption done more often than necessary. *Subject to cyclic restart.
		Resource Ordering.	*Feasibility of enforcing via compile-time checks. *Run-time computation not needed since problem is solved in System Design.	*Preemption done without much use. *Incremental resource requests disallowed.
AVOIDANCE	Selects mid-way between that of Detection and Prevention.	Manipulate to find at least one safe path.	*No Preemption necessary.	*Need to know future resource requirements. *Processes can be blocked for long periods.

TABLE 2.2: SUMMARY OF DEADLOCK HANDLING TECHNIQUES IN DISTRIBUTED DATA BASES

PRINCIPLE	ADVANTAGES	MAJOR COSTS	RELEVANT REF.
DETECTION	*Facilitates on-line approach. *Suits distributed data base environment fairly well.	*Run-time cost. *High rollback & restart costs. *Communication costs in maintaining information for deadlock detection.	*Isloor & Marsland *Goldman *Mahmoud & Riordon *Chandra
PREVENTION	*Rollback & restart is not necessary. *In the absence of software errors, consistency of data-base is guaranteed.	*Costs of waiting for highly data driven processes. *Communication costs in trans-mitting the advance informa-tion on future requests.	*Maryanski *Chu & Ohlmacher
AVOIDANCE	*Minimal use of rollback & restart. *Data resource allocation policy is mid-way between that of highly conservative prevention & very liberal detection principles.	*Costs of waiting processes & run-time costs. *Moderate rollback cost. *Communication costs of trans-mitting future data resource allocation requests.	*Chu & Ohlmacher

allocation still leaves at least one safe path for all the processes to complete their execution. Effectively, the scheme projects detection into the future in order to keep the system from committing itself to an allocation which will eventually lead to deadlock. To avoid deadlock, therefore, it is necessary to have some advance information on the future resource requirements of processes.

Empirical observations have suggested that, to a great extent, deadlock prevention mechanisms undercommit resources while the detection techniques give away resources so freely that deadlock or near-deadlock situations arise frequently. Avoidance schemes select a mid-way approach between highly conservative prevention mechanisms and very liberal detection techniques.

Avoidance schemes do not require preemption, but do need knowledge of the future. This knowledge may be in the form of upper bounds on the quantity of each resource group required by the process. If the upper bounds are generous, the resources are used inefficiently. Obtaining technically good upper bounds is difficult in the avoidance approach. In a heavily loaded system with most resources allocated, there are very few safe allocations for the outstanding requests. This may lead to processes getting blocked for long periods while holding useful resources.

In both prevention and avoidance of deadlock cases, recovery from systems implementation programming error needs

a rollback mechanism. A complete pictorial view of these basic strategies and their comparison is depicted in Tables 2.1 and 2.2.

Ignoring Deadlocks:

One less novel approach to handling deadlocks is to ignore them. Such strategies have been referred as an "indifferent strategy" [Isloor and Marsland, 1978] or a "no strategy strategy" [Holt, 1972]. In contrast to detection, prevention, or avoidance methods, ignoring deadlocks saves the overhead involved in the maintenance of algorithms. In such a case, the onus of recognizing deadlocks is borne by either a shrewd computer operator discovering blocked processes, or a skilled user waiting for an answer. Ignoring deadlocks can have disastrous effects on the consistency of data in both distributed and centralized database systems.

2.4 Models of Deadlock

In considering the deadlock problem, our interest in this section lies in the representation of process interactions while allocating resources to processes. Processes in computer systems can be dynamic in the sense that one process may create another. Processes are said to interact explicitly when they inter-communicate among themselves, using messages created by the sending processes and consumed by the receiving processes. Process

interactions as a result of competition for access to physical objects are termed implicit. Either explicit or implicit interactions can lead to the blocking of processes. Holt has proposed the distinction of "reusable resources" and "consumable resources" to model implicit and explicit interactions respectively [Holt, 1971, 1972].

The physical devices of a computer system, such as tape drives, disks, memory, channels, are reusable resources. There is always a fixed total number of units of these resources in any system. Any unit of a particular resource can be held by one process at a time. Thus, each unit of a resource is either free for allocation or is held by a process. The allocation strategies specify the unit of the resource. For example memory may be allocated by pages, disks may be held in units of tracks or entire disks, and data may be assigned as records, fields, entities, or files.

Message text from operators, external interrupts, inter-communication between processes, and card images produced by a card reader are examples of consumable resources. Typically, the total number of resource units is not fixed. When acquired by a process, every available unit of a resource ceases to exist. A process which creates a consumable resource must be treated as if it were holding the resource. Such a creator of a resource may release any number of units of the resource. Consumable resources are created and released by a producing process, and are requested, acquired and used by other processes. Whereas,

reusable resources are assigned by the resource manager to requesting processes which, after acquiring and using them, return them to the manager.

Many resources, such as tape drives and printers, permit only exclusive use by one process at a time but others, like data resources and read-only programs, may allow shared use by several processes. Another characteristic of a resource is the possibility of preemption. Some resources may be taken back by the system without any action by the process. Such a process is then either aborted, rolled back (if necessary) and restarted, or forced to rerequest and thus wait for the preempted resource. The cost of aborting or restarting the process accounts for the inherent losses due to preemption. In certain cases, it is possible to suspend the process and preempt the resource, yet preserve the current states of the process and its use of that resource for a later resumption. Such resumption does not lose processing time already spent. Typical examples of resumption are:

(a) CPU interrupts, in which the resource preempted is the CPU. The information that must be preserved for later restoration is the status of the process (for example registers and the "program status word"); and

(b) swapping, in which primary memory is the preempted resource and backup is provided in secondary storage.

When the system has different resource types and more than one resource of the same type, the degree of complexity

of the deadlock problem increases. Attempts to model and formalize the problem have resulted in two fairly interesting major proposals [Coffman et al., 1971; Holt, 1972]. In these proposals, process interactions have been represented by using graph-theoretic models, and deadlocks have been expressed precisely in terms of graphs.

Given a set of processes and a set of distinct resources in use by these processes, Coffman et al. define a state graph as a directed graph whose nodes correspond to the resources and whose edges are defined as follows: At any given instant of time, there exists an edge directed from a resource node R_i to a resource node R_j, provided some process P has access to R_i and has requested access to R_j. It has been shown that a directed circuit in the state graph is a necessary and sufficient condition for deadlock.

Example 2.1: Let P_1, P_2, P_3 and R_1, R_2, R_3 be respectively the processes and resources in the system. Let R_1 be held for shared access by both P_1 and P_2, and let P_3 be waiting for exclusive access to R_1. Assume that R_2 and R_3 are held for exclusive access by P_3 and P_1 respectively, while P_1 and P_2 respectively wait for exclusive access. The process interactions here can be represented by the state graph shown in Figure 2.3. The existence of a directed circuit involving three nodes in the state graph clearly means that there exists at least three deadlocked processes. For clarity and better understanding, we have labeled the edges in the state graph. For instance, the edge directed

from R_1 to R_3 is labeled sP_2e indicating that P_2 holds R_1 for shared access and is waiting for exclusive access to R_3, and so on.

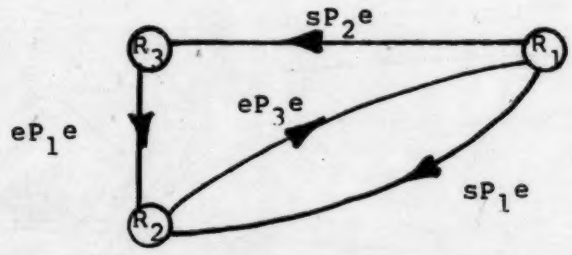


Figure 2.3: State Graph for Example 2.1.

For the general case, with more than one unit of some resources, the state graph defined above is inappropriate. Coffman et al. propose that the resources be partitioned into different types, in which resources of a given type are identical. The nodes in the state graph then represent resource types. A directed edge in the graph exists between a node representing one resource type to another, whenever any process has acquired access to at least one unit of the former resource type and has requested access to at least one unit of the latter type. A directed circuit in such a generalized state graph is still a necessity for deadlock existence. However, it is not sufficient.

Holt's model of a system of interacting processes is a classical piece of work, which is thorough and comprehensive. The characteristic of the approach is the use of a "general resource system" which models reusable as well as consumable resources. A general resource graph is defined

as a bipartite graph whose disjoint sets of nodes are the set of processes and the set of resources (reusable and consumable). The set of resources is associated with an available units vector whose elements are integers of the quantity of resources available. Edges directed from a process node to a resource node are termed request edges. Edges directed from reusable and consumable resource nodes to processes are called assignment and producer edges respectively. A process is blocked if and only if the number of request edges from this process to a particular resource exceeds the number of available units of the resource. A process is deadlocked when it is impossible to get the process out of the blocked state. In that approach, a "graph reductions" method is introduced to check if a process is deadlocked. A graph reduction corresponds to the best set of operations a particular process can execute to help unblock other processes.

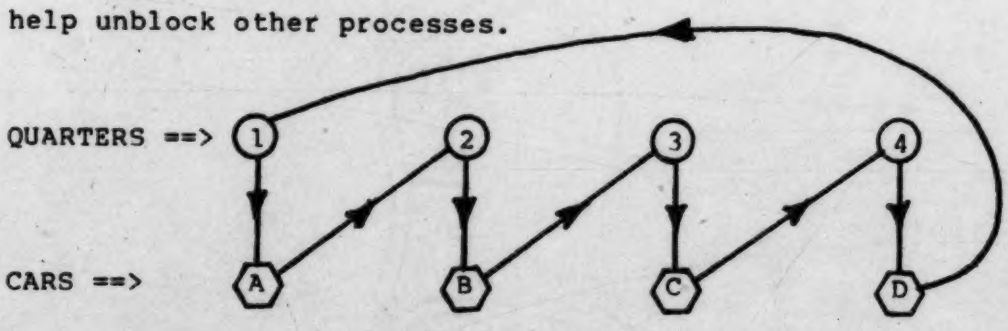


Figure 2.4: General resource graph for the traffic deadlock of Figure 2.1.

It is shown that for a general resource graph in which all processes having requests are blocked, the existence of a directed circuit is a necessary and sufficient condition for deadlock. It is further derived that for a general

resource system with single unit requests, a deadlock exists if and only if there is a cycle in the graph. A simple illustration of a general resource graph corresponding to the traffic deadlock of Figure 2.1 is shown in Figure 2.4 (in the situation when all the cars intend going straight).

It is evident from Figure 2.4 that the cars A, B, C and D are deadlocked. Intuitively, a necessary and sufficient condition for the existence of deadlock is that there exists a circular chain of processes in which each process holds exclusive and non-preemptible control of some resource, and is requesting access to at least one resource held by the next process in the circular chain of waiting processes.

2.5 Comparison and Contrast of Deadlock Problem in the Three Fields

Considerable research has been done on the deadlock problem in operating systems, and is surveyed in Section 2.6.1. Briefly, three broad categories of algorithms have been proposed:(i) detection, (ii) avoidance, and (iii) prevention. It has been shown that in certain cases, it is possible to suspend the process and preempt a resource, yet preserve the current states of the process and its use of that resource for a later resumption.

Even though the general principles for deadlock handling that have been developed for operating systems are applicable in database systems, several additional problems

arise. The resources which processes may wish to lock for exclusive use include pieces of shared data. Very often the processes may issue lock requests, the locking criterion for which depends on data values such as: "LOCK THE EMPLOYEE RECORDS OF ALL EMPLOYEES IN THE SYSTEMS PROGRAMMING DEPT." The complications introduced by this aspect are normally absent in an operating system environment. In addition, a particular data resource may be described in more than one way, or the nature of a data resource may be altered by a process operating on it. Locks may be interdependent in the sense that further locks may have to be requested depending on the first lock. Because of these complications many conventional approaches to resource locking in operating systems are made difficult or impossible in the database environment. In the case of preemption of a data resource, the necessity of maintaining uniformly correct data in the system dictates the abortion, rollback, and restarting of one or more processes. The cost of abortion and rollback is generally expensive and significantly complicates the deadlock problem.

An autonomous component behaviour is the main characteristic of operating systems for a network of computers, which greatly aggravates the control problems. Presence of appreciable time-lags renders synchronizing the actions of the various controllers in the system much more difficult. Moreover, the deadlock problem in distributed systems is somewhat different, since in geographically

distributed databases all information needed to detect deadlocks is not necessarily available at any single installation. Consider the example of Figure 2.2, but with two separate computers C_1 and C_2, as shown in Figure 2.5. Assume that the process P and resource entity M reside at C_1 and process Q and resource entity N reside at C_2 respectively. Processes P and Q, after updating local data resources M and N, arrive at remote locations and get involved in a deadlock which neither computer C_1 nor C_2 can detect, based on the information available at their respective installations.

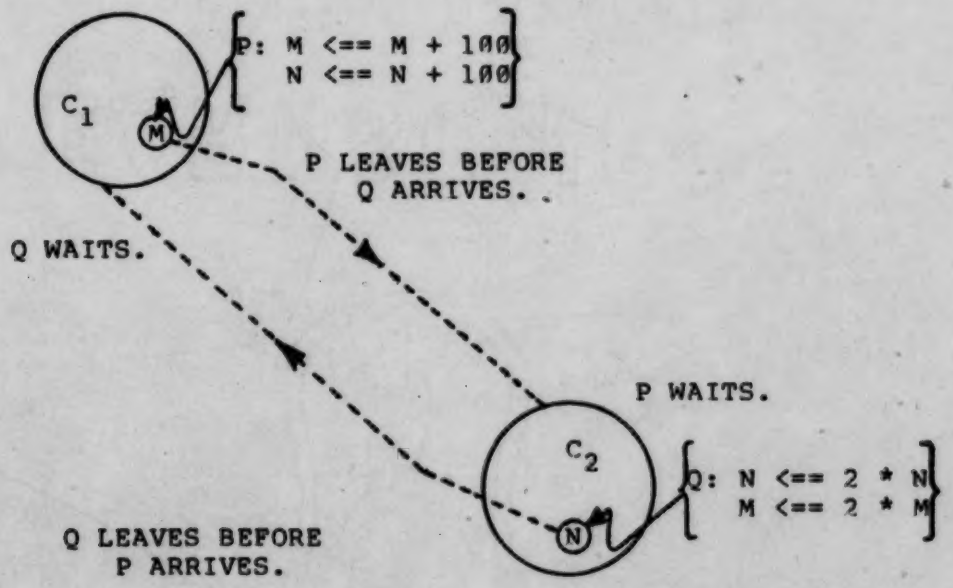


Figure 2.5: Distributed database deadlock: Example of Figure 2.2 in a distributed environment.

Communication delays may lead to synchronization problems in obtaining an accurate view of the status of the

computer network. Typically, an existing deadlock may not be detected, or a deadlock may be indicated where one no longer exists. Synchronizing the updates of multiple-copy files is non-trivial. Also, abortion, rollback and recovery become very involved and require further communication.

2.6 Survey of Deadlock Handling Schemes

Many different deadlock handling algorithms and approaches that have been proposed for operating systems, database systems and distributed databases are presented.

2.6.1 Operating Systems

A plethora of research papers has appeared on deadlocks in operating systems. The most notable among them are discussed below.

The approaches suggested by Havender[1968] exclude *a priori* the possibility of deadlock by putting certain constraints on the way in which requests for resources may be made. All the required resources must be requested and granted before the process can proceed. In a second strategy, when a process holding certain resources is denied a further request, the process must release all of its original resources and rerequest them together with the additional ones. The third strategy utilizes the principle of resource ordering. The capabilities of these three schemes have been discussed in Section 2.3.

The Bankers algorithm [Habermann, 1969] is a practical example of avoidance. The approach uses the "Maximum Claims Strategy" with regard to information about the future resource requirements for each process. This information is provided in the form of an upper bound on the quantity of each resource group that will be used by each process. A state is considered safe if there is a process which can run to completion using only the available resources and those already allocated to it. A state further derived from such a state is also safe. Deadlock avoidance is achieved by testing each possible allocation and making only those which lead to safe states. If the process which is the originator of this request can run to completion and release the resources it holds, then all other processes in the system can be completed, since the state previous to the request was safe.

Hebalkar[1970] uses a graph model to represent processes of more general structure (asynchronous systems) than those considered by Habermann. In that model, nodes represent transitions of a computation and edges represent the demand vectors of resources. A computation splits into parallel subcomputations, which can merge. The state of a system can thus be represented by a cut-set of a graph. Safe, as well as deadlock states can be represented in this model. Advance information in the form of demand vectors of resources in the graph model has been used to design algorithms to prevent deadlocks.

Coffman et al., use the model illustrated in Section 2.4 to provide a deadlock detection and recovery scheme for the case of more than one resource of a given type. Such a method needs a more elaborate state description mechanism than a state graph, and is supplemented with "allocation" and "request" matrices and an "available resources vector". An algorithm is designed which uses these data structures to discover a deadlock by simply investigating every possible sequence of remaining processes to be completed. This algorithm precisely identifies the complete set of deadlocked processes, and runs in time proportional to the square of the number of processes.

A further algorithm [Coffman et al., 1971] for deadlock recovery is suggested, by preemption of a subset of resources that would incur a minimum cost. An efficient branch-and-bound tree-search technique is utilized to find a minimum cost solution. This algorithm facilitates the inclusion of preemptible resource types of varying preemption costs in the notion of deadlock. Thus, the recovery technique is designed to avoid preemption of resources with high inherent losses. An informal discussion of certain theoretical aspects of deadlock avoidance using information on resource requirements is also provided.

A comprehensive proposal for deadlock handling [Holt, 1972], is dealt with in detail in Section 2.4. Holt suggests an efficient deadlock detection algorithm for the special case of general resource systems with single unit

requests. The algorithm tests the general resource graph for the existence of a cycle. This is achieved by determining if the successive elimination of all sink nodes produces predecessors which are also sinks. A sink node is one with no edges emanating from it (i.e. no wait requests). All nodes will become sinks if and only if the graph is devoid of a cycle. The mechanism is a simplified version of successive graph reduction.

An effective algorithm [Holt, 1972] is used to determine if a particular blocked process is deadlocked. The algorithm systematically traces out all paths leading to the corresponding process node. A path which leads to a sink exists if and only if the process was not deadlocked. Both of these algorithms execute in time proportional to the number of edges in the graph. A weighted edge is used to represent allocation of multiple units of a resource to a process.

The relationship of graph reducibility in consumable resource systems and the security from deadlock of such a system is investigated. A deadlock detection algorithm for reusable resource systems is presented. The technique of graph reductions is used to improve Habermann's original deadlock prevention algorithm. The importance of graph reductions in investigating the deadlock problem is emphasized [Holt, 1972].

2.6.2 Database Systems

Various aspects of concurrent operation of database processes have been the topics of active research in the recent past. Several reports have dealt with the deadlock problem in database systems, the notable among them being surveyed below.

Shoshani and Bernstein[1969] assume that a database can be represented by a graph, with nodes representing collections of information. A directed edge exists between two nodes, if one node contains the address of the other. Such a database is assumed to be accessed by a set of routines called primitives. Under concurrent operation, two or more primitives may conflict with each other while accessing a node. To overcome these conflicts a procedure called the "walking algorithm" was formulated. It requires that (i) a primitive lock the next node it wishes to access before unlocking the node it is currently accessing, and (ii) it keep a node locked only during the time of its accessing. Further, a method called the two-step algorithm is developed to overcome the drawbacks (including deadlock avoidance) of the walking algorithm.

The LOCK-UNLOCK mechanism of the CODASYL approach to data management, which enables incremental allocation of data resources to processes, is described by King and Collmeyer[1973]. A database access state graph is used to define the state of all accesses to a database. Thus the

LOCK-UNLOCK mechanism can be modeled by state transition functions that map access state graphs to access state graphs. The operations of the functions LOCK, UNLOCK, ALLOCATE and DEALLOCATE are modeled. Further, it is shown that the ALLOCATE function is the only one that can precipitate a deadlock. A necessary and sufficient condition for the existence of a deadlock is derived, in terms of the effect of the ALLOCATE function. A scheme is devised to detect a deadlock using the fact that one can occur only when an allocation request results in a process being blocked. In the event of a deadlock, a recovery scheme is suggested. A major shortcoming in the approach is that a process cannot have more than one outstanding request. We have stated in Chapter 3 that in real world applications it is a very common necessity for a process to have at one time more than one outstanding request.

One proposed technique [Chamberlin et al., 1974] is a shrewd modification and combination of: (i) trying to preclaim needed resources; (ii) if preclaiming data resources leads to a deadlock, preempting data resources; and (iii) imposing a presequencing scheme for processes by time stamping, to avoid deadlock due to indefinite delay. The method requires each process to lock all of its data resources during a "seize phase", before starting the "execution phase". The concurrent running of the seize phases of processes raises the deadlock problem. During the seize phase, preemption from a process of locked resources

is possible, and backing-up a process to the start of its seize phase is easy. Once in its execution phase, a process is not allowed to claim more resources. The end of an execution phase is first signalled by the release of all resources held, followed by the possible starting of a new seize phase. An age indicator attached to the processes is used to avoid deadlock due to indefinite delay of processes, thus the method is deadlock-free. A formal proof of correctness of the deadlock-free property of the scheme has also been demonstrated. This algorithm makes the dynamic resource allocation based on previous/current calculations ineffective.

One-level lockout and two-level lockout mechanisms [Schlageter, 1975] are considered for synchronizing database access. In the one-level lockout scheme, the readers can access the database at any time, regardless of the allocation state, whereas the writers are required to lock the data resources. In the two-level lockout scheme, readers may share the same data, but have the right to prevent writers from accessing this data. This is implemented by using primitives LOCKR (read) and LOCKW (write). Data locked by LOCKR can be accessed by any reader. Data locked by LOCKW can be accessed by readers which do not need to be protected against changes of data. Presence of a cycle in the process graph is shown as a necessary and sufficient condition for deadlock existence. Given the process graph, the deadlock detection procedure

starts at the blocked process node and tests if a path reaches the process node again. Under the two-level lockout mechanism, starting at a blocked process node and traversing paths to detect deadlocks is no longer simple.

One scheme proposed for deadlock avoidance in database systems [Lomet, 1977] requires the processes to pre-declare their anticipated resource requirements, with the system granting only safe requests. The algorithm is tailored to the needs of database systems, unlike the approaches of Habermann[1969], and Holt[1972]. A series of time-varying graph representations is defined for database interactions. A "holds-claims graph" represents only those processes that are currently making a claim on some common system resources. A "claims-claims graph" represents the processes which are potentially capable of denying resources to one another, their claims being in contention. A "holds-holds graph" represents the allocation status of the system. A deadlock exists if and only if there is a cycle in the holds-holds graph, whereas a deadlock can be avoided if and only if the holds-claims graph is cycle-free. A deadlock avoidance scheme has been devised which performs appropriate actions on claims-claims, holds-claims, and holds-holds graphs in the event of process entry/deletion, and resource request/release. Even though the possibility of indefinite delay is not entirely eliminated, it is reduced as a result of the strategy of incrementally granting requests.

2.6.3 Distributed Databases

Relatively little work has been reported on deadlock in distributed systems. A thorough discussion of all earlier attempts to handle deadlocks in distributed databases, and their drawbacks, is provided. A new approach suggested by us is dealt with in detail in Chapter 3.

Prevention of deadlocks in distributed databases has been the subject of papers by Chu and Ohlmacher[1974] and Maryanski[1977]. In their first approach Chu and Ohlmacher require that all data resources be allocated to the processes before initiation, which in turn needlessly delays the processes. Their second technique is based on the concept of a process set, which is a collection of processes with access to common data resources. A process is allowed to proceed only if all data resources required by the process and the members of its process set are available. In Maryanski's proposal, each process has to communicate its shared data resources list (conceptually similar to a process set) to all other processes before it can proceed. This shared data resources list is determined by using what is called a process profile, which contains a list of data resources that can be updated by the process. The communication and computation of process sets (or shared data resources list), which is performed continually as processes enter and leave the system, makes heavy demands on the system.

Techniques for deadlock detection in a network environment [Chandra et al., 1974; Mahmoud and Riordon, 1976, 1977; Goldman, 1977] have been proposed. At each installation, Chandra et al. require the maintenance of a resource table which contains information pertaining to local resources allocated to processes, processes waiting for access to local resources, remote resources allocated to local processes, and local processes waiting for access to remote resources. The type of access requested by the process is also stored. They hypothesized that by using such tables, well known algorithms for detecting deadlocks in a single system could be extended to detect deadlocks in a network of computers, by communication between installations and appropriate expansion of resource tables. Schemes to expand resource tables in a network environment were included. However, Goldman has shown an example in which a deadlock is not detected by their proposed scheme.

The Centralized Control approach to deadlock detection in distributed databases of Mahmoud and Riordon creates an overall picture of the global network status by using file queues and pre-test queues (a queue of requests which can only be granted at a future time) received from all other installations in the network. As the identifiable unit of object-data becomes smaller in size, message congestion at the control node increases to degrade the network performance. The authors also propose a Distributed Control approach in which, in a network of 'n' computers, each

installation transmits (n-1) identical messages containing status and queues of files. Each installation thus receives (n-1) different messages. As shown by Goldman this approach has a flaw, in which a certain deadlock goes undetected. In any case, all these proposals require the communication of large tables between installations.

The Detection schemes of Goldman are based on the creation and expansion of an Ordered Blocked Process List (OBPL). An OBPL is a list of processes in which each process except the last one is waiting for a data resource held by the next process in the list. Whenever an OBPL is transmitted between installations, a data resource name is inserted into the "single data resource identification part" of the OBPL. The last process in the list either has access to, or is waiting for, that resource. In the former case the state (blocked or active) of the last process in the OBPL must be determined, while in the latter case one needs to know the state of the process which holds the data resource. Goldman proposes techniques to determine these states and to eventually detect deadlock (if any).

Even though Goldman's method seems sound, it does have some shortcomings. For instance, no process may have more than one outstanding resource request, which is not generally the case in real world situations, as illustrated in Appendix B. Also, when several readers share access to a data resource, Goldman requires the creation and expansion of one different copy of the OBPL for each reader; since if

one of the readers is deadlocked, then so is any process which requests access to that resource. It is possible that OBPL's, while undergoing expansion, could be transferred (sequentially) among several installations or several times between the same two installations before a deadlock is detected. Furthermore, OBPL's could become large, leading to substantial overhead, especially when records or entities are considered as data resources instead of files.

The primary disadvantage of all the existing methods is that they cannot recognize that deadlock is imminent without substantial communications between the computers in the network. Thus, the algorithms described above cannot be used effectively for on-line detection since they are very susceptible to "synchronization error", in which either a deadlock is indicated where one no longer exists or a deadlock occurs and is not recognized - when two autonomous computers concurrently allocate resources before advising each other of their actions.

2.7 Combined or Mixed Approach to Deadlock Handling

It is hypothesized [Howard, 1973] that none of the three basic approaches alone - detection, avoidance, prevention - suits the complete set of resource allocation problems in operating systems. Instead, different subproblems of resource allocation can be optimally handled by different individual techniques. At the same time, all the techniques can cooperate globally to prevent deadlocks.

The basis for the method lies in the hierarchical structure of operating systems. Resource ordering in a hierarchical structure provides the framework for a mixed technique. In many well-designed operating systems, several levels of software are provided. Each level modifies and extends the capabilities provided by the underlying level. The implementation of an operation in a higher level is achieved by creating a lower level process. Such a created process performs the actions, obtains the results and indicates its completion through a message to the higher level process. Such messages are treated as consumable resources by deadlock handling mechanisms. In the case of resource ordering, this implies that a higher level process cannot hold any resources required by a lower level process it creates. Such a restriction automatically enforces a "natural" ordering of resources in a hierarchical system. Resources required by lower level processes should appear later in the ordering. The ordering is also applied to classes of resources.

Howard provides the following example involving the CDC 6600 in use at the University of Texas at Austin. The system resources are classified into the following categories:

(i) space in the swapping store;

(ii) assignable devices such as tape drives, and job resources such as access to files;

(iii) central memory for user jobs; and

(iv) internal resources such as memory for transient system overlays, and channel and controller access.

A combined approach to the deadlock problem in the above system works as follows: Preallocation of the maximum requirement for each process seems to be the most practical method for the swapping space. For the job resources, avoidance is the most reasonable solution since it is usually possible to obtain considerable information on the future resource requirements of a job from its control cards. Detection and preemption are not necessarily suitable due to possible file updates. Prevention using preemption is the logical preference for central memory, provided the system is capable of swapping. Detection is also a possibility, but is not recommended if prevention is also possible. Prevention in the form of resource ordering is the best choice for internal system resources. The hierarchical nature of internal system structures usually ensures a natural resource ordering. Even though, theoretically it is preferable to use a single algorithm, a combined approach has practical advantages in its favour.

A comprehensive combined approach to deadlock handling in database systems or distributed databases has not been devised so far. However, the idea of mixed solutions has been applied in our approach (Chapter 3) to on-line detection in distributed databases, when there are multiple outstanding requests on a data resource released by a completing process. In this case of a deadlock-free system,

an indiscreet resource allocation decision can potentially lead to deadlock. We will be showing that there exists at least one process in the set of waiting processes such that the allocation of the resource to this process maintains the system deadlock-free. In other words, a potential deadlock is detected and avoided accordingly. The approach utilizes both the detection as well as the avoidance principles.

2.8 Deadlock Problem vis-a-vis Game Playing

The deadlock avoidance problem has been informally defined as: "From some a priori information about the processes, the resources, the operating system, etc., determine what situations may be realized without endangering the smooth running of the system" [Devillers, 1977]. These situations are termed safe, and the others unsafe.

The game-playing model of the deadlock situation has not received much attention, perhaps because it appears to be practical only for small problems. Nevertheless it does provide a valuable alternative viewpoint. The general approach taken by Devillers[1977] is to recognize that the resource allocation problem is effectively a zero sum game in which resource manager (RM) is competing against independent processes demanding service. In this case the manager may be thought of as winning if all the processes complete successfully, while the opposition wins if they create a deadlock. A flow chart model which in some sense

generalizes Habermann's Maximum Claims model and Hebalkar's Task Step model, is proposed. Unfortunately with this model it is not easy to define the safe states, not only because a process may have more than one possible series of steps that it may traverse (future history), but also because there may not exist a simple worst possible future history for each process. To overcome this problem Devillers proposes a global approach. During the execution of a system the processes pass through various states. For the deadlock problem three types of states are considered.

(a) The "working state" representing the state of being in some process step.

(b) The "transit state" in which some process has completed one step and is waiting the resource manager's permission to enter the next step of the process.

(c) The "terminated state" which represents a process that has completed all the steps in its task.

The states of the system can thus be characterized by the status of the various processes and the identity of the player making the move. Every such configuration is associated with a resource need. If this need exceeds the availability of some resource types, an "unattainable" configuration and set of states will result.

The moves of the players are dependent on the philosophy of the RM. The RM may detect at most one, at least one, or any number of requests for transition, and may grant at most one or any number of requests in one move.

Corresponding games can be shown which are equivalent to the deadlock avoidance problem. The possible future evolutions of the system can be represented by the "graph of this game", where the nodes are the states and the edges are the moves. This is a finite graph with an infinite associated game tree. However, it is relevant to note that:

* The strategy (deterministic or probabilistic) of the processes may not induce infinite games;
* The game stops when a terminal node (no successors) is reached;
* The RM wins if the attained terminal node corresponds to the process completion;
* The RM loses if the states attained are without attainable successors, i.e. where none of the processes is in a working state; and some are in transit states, but no requests may be granted without producing unattainable configurations.

A state is defined <u>safe</u> if and only if, from this state, a strategy exists for the RM which ensures its success whatever strategy the processes choose. A state will be <u>unsafe</u> if and only if it is not safe. A state will be <u>losing</u> if and only if, from this state, a strategy exists for the processes such that the RM will lose the game whatever strategy it chooses. Devillers presents an algorithm for the construction of the set of unsafe states. This approach throws new light on the deadlock problem, and the notion of risk is explicitly formalized. It also

provides a basis for a systematic study of the properties of the safe states.

2.9 On Probabilistic Models of Deadlock

The approach to the deadlock problem taken by Ellis [1974] is different from all previous approaches in the sense that probabilistic techniques are used to investigate the likelihood of deadlock occurrence in certain classes of computer systems. Any system state diagram used to represent process-resource interactions can also be viewed as a finite state automaton, provided that the number of states is finite. A probability measure can be attached to an occurrence of each possible transition out of a given state. The sum of the probabilities associated with transitions out of a given state is required to be unity. Ellis assumes a random resource allocation model, which forms a Markov chain. By adding an auxiliary storage to the automaton, first-come-first-serve (FCFS) and last-come-first-serve (LCFS) schedulers can be modeled to form, respectively, a probabilistic queue automaton and a probabilistic push down automaton. The probability of deadlock is measured in terms of the expected value of (i) the number of system actions to deadlock or (ii) the number of resource allocations to deadlock. Calculations are carried out for systems containing small numbers of processes and resources. For a system with 2 resources and 2 processes the mean time to deadlock under FCFS or LCFS

scheduling is shown to be slightly less than that under random allocation. The fact that one would intuitively expect the probability of deadlock to decrease if the number of units of the resources increases while the number of processes remain fixed, is substantiated. Conversely, in a fixed resource system, increasing the number of processes increases the probability of deadlock since more processes would be competing for the same number of resources. However, it is not intuitively clear what happens to the deadlock probability if the number of processes and resources are uniformly increased. For small systems Ellis has shown that the probability of deadlock actually increases. However, since the model considered no more than 5 resources and processes, which is by no means a very large one in the commercial world, more research is clearly needed in that area.

2.10 Current Implementations of Deadlock Handling Schemes

The internal structure of the system-wide shared-file table in the Michigan Terminal System (MTS) is explained in detail here. MTS is a major operating system under continuous development by several major universities in various countries. Before any specific file operation is performed, the files are locked in MTS in one of three inclusive levels (read, update, or destroy). In order to ensure that the rules of concurrent usage are not violated before locking, MTS maintains a table indicating at any

instant,

* all the files currently open and/or locked,
* how they are locked and by what task,
* what tasks are currently waiting to lock a file, and how they are waiting.

This table facilitates determining whether or not a particular type of opening and/or locking of a file can be allowed, in accordance with the following rules of concurrent usage:

(i) If a file is not locked for updating or destroying, any number of tasks can have this file locked simultaneously for reading;

(ii) If a file is not locked for reading or destroying, then only one task can have this file locked for updating (writing, emptying or truncating); and

(iii) If a file is not open, and not locked for reading or updating, then only one task can have this file locked for destroying (or renaming or permitting).

If a file cannot be locked as requested, the task is queued to wait on the file. Waiting can lead to deadlock due to mutual blocking of tasks. Blocking is defined as follows:

(a) a task waiting to destroy a file is blocked by a task with the file open;

(b) a task waiting to update or destroy a file is blocked by a task with the file locked to read;

(c) a task waiting to read, update, or destroy a file is

blocked by a task with the file locked for update; and (d) a task waiting to open, read, update, or destroy a file is blocked by a task with the file locked to destroy.

The method that MTS employs to detect a deadlock involving multiple files is to define a relation BLOCKS, where a Task T1 BLOCKS Task T2 if and only if Task T1 has a file open and/or locked in such a way that Task T2 is blocked from using that file. An M x M matrix, representing the relation BLOCKS, is constructed. M is the total number of tasks, with files open and/or locked, blocking another task or being blocked. The transitive closure (BLOCKS+) of the relation BLOCKS is computed by using Warshall's algorithm [Warshall, 1962]. A deadlock situation exists when Task T BLOCKS+ Task T is true.

Data management systems, on the other hand, vary in the way they detect and resolve a deadlock. As has been indicated elsewhere, many of the early systems implemented techniques which were quite rudimentary. In some systems, such as IDMS (Integrated Data base Management Systems marketed by Cullinane Corp.), and TOTAL (CINCOM, Inc.) deadlock is not possible because of restrictions on processes. In these systems a process can lock only one record at a time, and so heavy restrictions are placed on users to ensure that deadlocks will never occur. Consequently, they are neither general nor realistic.

In the data management system ADABAS (Adaptable DAta

BAse System, a product of Software AG, West Germany), if a process requests a locked record five times, the record is unlocked. The requesting process then gets hold of the record and proceeds. Quite often a process, after locking the record, may attempt to update it, only to find that it had lost control of the record. The record will have to be reread and the update redone. This method leads to unnecessary preemptions and is also subject to a peculiar situation in which two processes may perpetually get involved in a cycle of locking, losing control before updating, and relocking the record concerned, as outlined in Section 2.3 and Appendix A.

CHAPTER 3

'ON-LINE' DEADLOCK DETECTION IN DISTRIBUTED DATABASES

3.1 Graph-Theoretic Model and Concepts

Certain important concepts with direct relevance to our algorithms are defined below. We choose a graph-theoretic deadlock model [Holt, 1972], proposed for operating systems, and extend it to represent process interactions in a distributed database.

The set of data resources (typically files, fields, records, or entities), represented by $\underline{D} = \{D_1, D_2, \ldots, D_m\}$, is held by a set of processes denoted by $\underline{P} = \{P_1, P_2, \ldots, P_n\}$, running concurrently (intuitively, processes initiated so far, but not completed) in a network of computers. A directed graph with nodes corresponding to each process and each data resource in $\underline{P}$ and $\underline{D}$ respectively, and with edges between nodes representing process interactions in the system, can be used to depict the system status. We formalize this in,

Definition 1: A system graph $G_s = (N,E)$ is a directed

bipartite graph[1] whose disjoint sets of nodes are those corresponding to P and D, respectively called process nodes and data resource nodes, such that $N = P \bigsqcup D$. An edge directed from a data resource node D_i to a process node P_j, denoted (D_i,P_j), is called a resource-process access(RPA) edge which specifies that the data resource D_i is held by process P_j. Similarly, a directed edge from a process node P_r to a data resource node D_s, denoted (P_r,D_s), is called a process-resource wait(PRW) edge which indicates that process P_r is waiting for access to data resource D_s. E is the union of the sets of RPA edges and PRW edges. The type of access granted to an RPA edge or requested by a PRW edge is indicated by the letter "e" or "s" for exclusive or shared access.

Remarks: A process cannot hold a data resource and be simultaneously waiting for access to it (i.e., cannot be self-blocking). It is, thus, necessary for the process to declare its most restrictive use of a data resource, in order to prevent the process from getting blocked waiting for exclusive access when it already has shared access. This implies that if (P_i,D_j) is an edge in the graph, then there does not exist an edge (D_j,P_i) and vice versa.

Definition 2: The reachable set of a node N_j of the system

1: For the definitions and understanding of graph-theoretic terms refer: Deo, N., Graph Theory with Applications to Engineering and Computer Science, Prentice Hall, Inc., Englewood Cliffs, N.J., 1974.

graph G_s, denoted by $R(N_j)$, is the set of all nodes in G_s such that there exists a path directed from N_j to all nodes in $R(N_j)$.

The notion of reachable set first introduced by Holt is used effectively in deadlock detection. As soon as the information on the addition or deletion of an edge to G_s is received, the reachable sets can be updated to detect deadlocks on-line, as proposed in this chapter, which do not need the transmission of large tables between installations.

A deadlock situation may arise when the following necessary conditions hold:

(a) the processes request exclusive control of the data resources (for updating);
(b) the processes hold data resources allocated to them, and wait for additional ones to run to completion; and
(c) the preemption of data resources from processes cannot be done without aborting the processes.

We formally characterize this in,

Definition 3: A state of deadlock in a "circular wait" condition is said to exist when, in a circular chain of processes, each process holds one or more data resources and has requested access to at least one data resource held by the next process in the chain.

3.2 A Running Example

In Figure 3.1, $\{P_0, P_1, P_2\}$, $\{P_3, P_4\}$, $\{P_5, P_6\}$, $\{P_7, P_8\}$

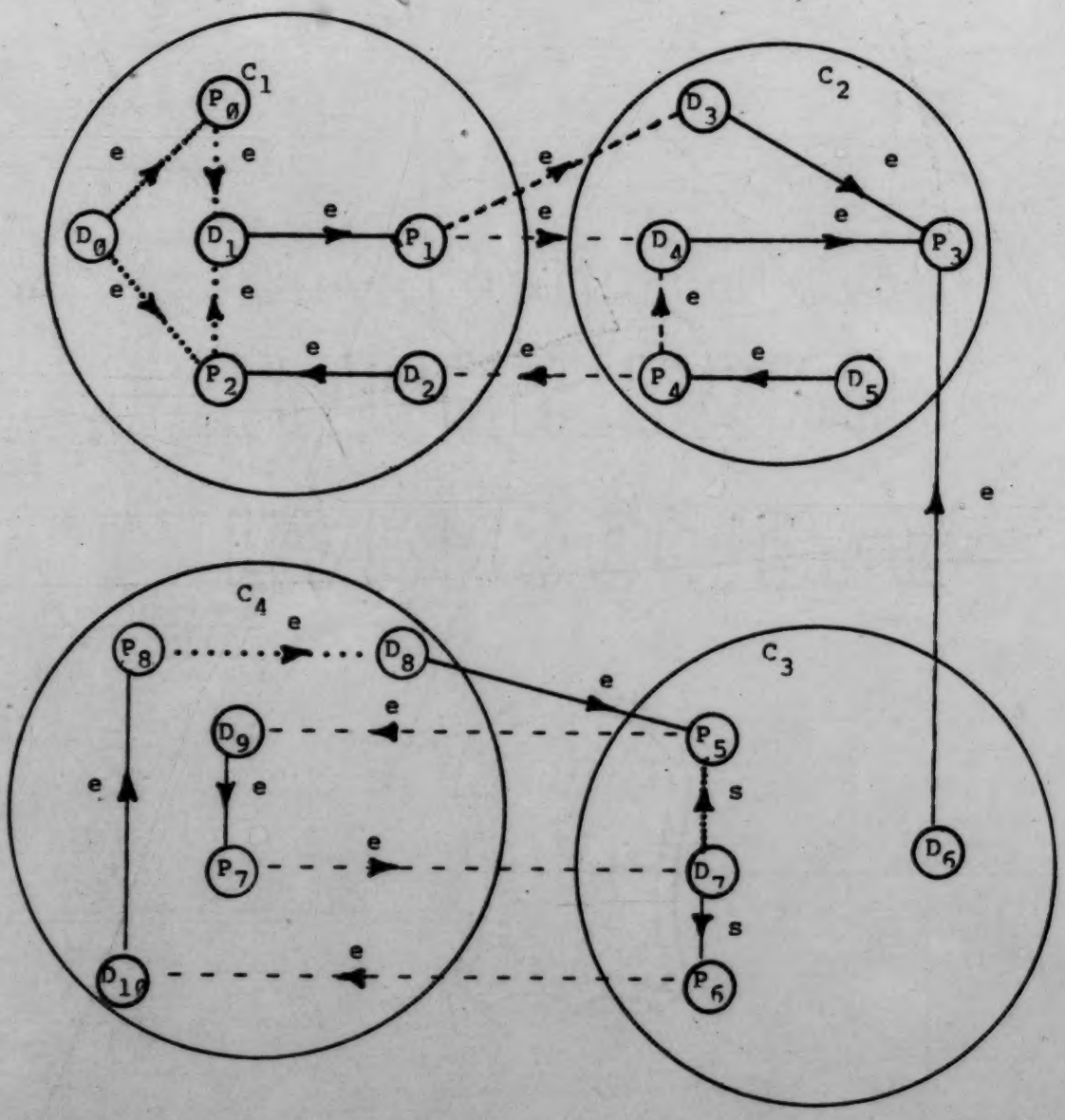


Figure 3.1: A system graph for a network configuration with four computers, C_1, C_2, C_3, and C_4, a set of concurrent processes $\{P_i\}_{0 \le i \le 8}$ and a set of data resources $\{D_j\}_{0 \le j \le 10}$.
Key: - - - - - PRW Edges,
——— RPA Edges.

are subsets of processes that exist in computers C_1, C_2, C_3 and C_4 respectively. Data resources $\{D_0, D_1, D_2\}$, $\{D_3, D_4, D_5\}$, $\{D_6, D_7\}$, $\{D_8, D_9, D_{10}\}$ reside at C_1, C_2, C_3, C_4 respectively. The RPA edges are shown by solid lines, the PRW edges by dashed lines. The closely dotted and the sparsely dotted lines represent RPA and PRW edges that have not yet been introduced into the system. These future requests are included to illustrate the various aspects involved in on-line deadlock detection. At computer C_2 of the network P_3 holds D_3, D_4, and D_6, and is active. P_4 holds D_5 and is waiting for access to D_4 and D_2, and hence is blocked. P_1, P_4, P_5, P_6 and P_7 are blocked. P_2, P_3 and P_8 are active. The reachable sets of nodes D_6, P_4, and P_2, for instance, are $\{P_3\}$, $\{D_4, P_3, D_2, P_2\}$ and $\{\phi\}$ respectively. If we assume that P_6 holds D_7 for shared access, then any request by P_5 for shared access to D_7 (which results in the introduction of the RPA edge from D_7 to P_5) will cause P_5 and P_7 to be deadlocked.

3.3 Necessary and Sufficient Conditions for Deadlocks

We have defined the concepts of system graph and reachable sets in Section 3.1; these are used in this section to develop necessary and sufficient conditions for the existence of deadlocks.

Let P_j be a process. If P_j belongs to its own reachable set, then there exists a path in G_s which starts at P_j and ultimately ends in P_j. Since G_s is a bipartite

graph, the edges in this path should be alternately selected from the two sets--PRW and RPA edges. This in turn means that process P_j is waiting for a data resource, which is held by another process, and this process in turn is waiting for a data resource held by the next one in line, and so on, till some process in the chain is waiting for the data resource held by P_j. Intuitively, each process in the circular chain holds the data resource for which the previous process in the circular chain has a waiting-access request, and is by itself waiting for the data resource held by the next process in the chain, leading to a circular wait condition. Thus if a process node P_j belongs to its own reachable set, then process P_j is deadlocked. And conversely, if a process P_j is deadlocked, the reachable set of the process node P_j contains the process node P_j. This provides us with a necessary and sufficient condition for the existence of deadlock. We register this fact in,

Theorem 1: A process P_j is deadlocked in a circular wait condition if and only if the reachable set of the corresponding process node in G_s contains the node itself. That is, process P_j is deadlocked in a circular wait condition if and only if process node $P_j \in R(P_j)$.

Proof:

(<==) $P_j \in R(P_j)$ implies the existence of a path ρ (say) in G_s, starting at P_j and terminating in P_j. Since G_s is a bipartite graph, the edges in ρ alternate between the two sets of edges: PRW and RPA edges. Without loss of

generality, let ρ be made of the edges (P_j, Dj_1), (Dj_1, Pj_1), (Pj_1, Dj_2),..., (Pj_{k-1}, Dj_k) and (Dj_k, P_j), where Dj_i for $1 \leq i \leq k$ are data resources and Pj_i for $1 \leq i \leq k-1$ are processes. Path ρ indicates that P_j is waiting for access to Dj_1 held by process Pj_1, each of the processes Pj_{i-1} is waiting for Dj_i held by the process Pj_i for all i, $2 \leq i \leq k-1$, and Pj_{k-1} is waiting for Dj_k held by P_j. In other words, each process is waiting for the next process to run to completion in the circular chain of waiting processes.

(==>) Let us assume that $P_j \notin R(P_j)$, implying that there does not exist a path starting at P_j and terminating in P_j. Thus, there must exist a linear ordering of the processes with the following property: P_i precedes P_k if there is a path from P_i to P_k in G_s. Without loss of generality, let such a linear ordering starting with P_j be P_j, Pj_1, Pj_2,..., Pj_s, implying that P_j is waiting for access to a data resource held by Pj_1 and Pj_{i-1} is waiting for access to data resource held by Pj_i for all i, $2 \leq i \leq s$. Thus, Pj_s can run to completion. Pj_{i-1} can therefore, run to completion as soon as Pj_i terminates for all i=s,s-1,...,3,2. Ultimately, P_j can run to completion when Pj_1 terminates, which is a contradiction. ■

If P_i and P_j are any two processes such that their reachable sets in G_s include both P_i and P_j, then the reachable sets of both should be same. Intuitively, if

there is a path from P_i to P_j, and vice versa, then all the nodes that are accessible from P_i are also accessible from P_j, and vice versa. In Corollary 1 we arrive at a necessary and sufficient condition to identify all processes involved in a deadlock.

Corollary 1: A set of processes $\{Pj_i\}_{1\le i\le k}$ is deadlocked in a circular wait condition if and only if $Pj_i \in R(Pj_i)$ for all i, $1\le i\le k$, and $R(Pj_1) = R(Pj_2) = \ldots = R(Pj_k)$.

Proof:

(<==) Since $Pj_i \in R(Pj_i)$ for all i, $1\le i\le k$, and the reachable sets of all processes are the same, by Theorem 1 the set of processes is deadlocked.

(==>) Since each process is deadlocked, we have by Theorem 1, $Pj_i \in R(Pj_i)$ for all i, $1\le i\le k$. Due to the circular wait condition, each process node in $\{Pj_i\}_{1\le i\le k}$ is accessible from the others.

That is, $R(Pj_i) \supseteq \bigcup_{\substack{1\le s\le k \\ s\neq i}} R(Pj_s)$ for all i, $1\le i\le k$.

Hence, $R(Pj_1) = R(Pj_2) = \ldots = R(Pj_k)$. ■

Remarks: Theorem 1 and Corollary 1 have similarities with the necessary and sufficient conditions for deadlock derived by Holt[1971] for "reusable resource graph with single unit resources". The membership of a process node in its own reachable set is equivalent to the existence of a cycle in Holt's reusable resource graph.

Corollary 1 points out the simplicity of recognizing

the set of processes involved in a deadlock. In Corollary 2, we show a sufficient condition for a process not deadlocked, to be blocked indefinitely by virtue of its waiting for a deadlocked process. Consequently, such a process is blocked forever.

Corollary 2: A process P_j such that $P_j \notin R(P_j)$ in G_s, is blocked forever if $R(P_j) \supseteq R(P_i)$ for any process P_i such that $P_i \in R(P_i)$ in G_s.

Proof: $P_i \in R(P_i)$ and $R(P_j) \supseteq R(P_i)$ implies that P_j is either waiting for P_i to run to completion, or there is a sequence of processes from P_j to P_i each one of which is waiting for the next one in the sequence to run to completion. In either case P_j cannot run to completion since P_i is deadlocked in a circular wait condition. ■

For instance, in the system graph of Figure 3.1, the introduction of a PRW edge from P_3 to D_5, consequent to P_3 requesting access to D_5, causes P_3 and P_4 to be deadlocked. P_1 is blocked forever but not deadlocked, since it waits for P_3 to release D_3 and D_4. The reachable set of P_1 is $\{D_2, D_3, D_4, D_5, P_2, P_3, P_4\}$ which contains the reachable set $\{D_2, D_4, D_5, P_2, P_3, P_4\}$ of the deadlocked processes P_3 and P_4.

3.3.1 Characteristics of Waiting Processes

In on-line algorithms for deadlock detection, the basic operation is to update the reachable sets every time a new edge is introduced in G_s or deleted from G_s. The introduction of a new edge, if it causes deadlock, implies

that the entering edge prompted a chain of processes to wait for each other. A process waits for other processes if the reachable set of the corresponding process node is non-null. Conversely, a process node with a non-null reachable set has the corresponding process waiting for at least one other process. We characterize this property of waiting processes in Lemma 1.

Lemma 1: A process P_i waits for a process P_j if and only if the process node $P_j \in R(P_i)$ in G_s.

Proof:

(<==) Since $P_j \in R(P_i)$, there exists a path from P_i to P_j. Let such a path (without loss of generality) be (P_i,D_{i_1}), (D_{i_1},P_{i_1}), ..., $(P_{i_{k-1}},D_{i_k})$, (D_{i_k},P_j). P_i is waiting for the data resource D_{i_1} held by P_{i_1}. Process $P_{i_{s-1}}$ is waiting for data resource D_{i_s} held by process P_{i_s} for all s, $2 \leq s \leq k-1$. $P_{i_{k-1}}$ is waiting for data resource D_{i_k} held by P_j. Thus, P_i is waiting for P_j.

(==>) P_i waits for P_j implies that either there is a data resource which is held by P_j for which P_i has an access request or there is a sequence of processes from P_i to P_j such that each process at least holds one data resource and is waiting for access to the data resource held by the next process in sequence. This in turn means that there exists a path from P_i to P_j. ■

Remarks: A process P_i waits for no data resource if $R(P_i)=\{\phi\}$, and thus, can run to completion. At any given time, a process is active and not blocked if and only if the

reachable set of the corresponding process node is null. Any process with a non-null reachable set is blocked, waiting for at least one process.

3.4 Deadlock Detection in Distributed DBMS

For the algorithms proposed in this chapter, all the necessary information to detect deadlocks is made available through a system graph at each installation. Maintaining the system graph is trivial, and requires communication only for processes and resources which are global in nature. For processes which are local, accessing only those local resources which have no global interactions of any kind, communication is not necessary. It is believed that for transaction processing systems over 95-99% of processes fall into this category[2]. However, to maintain a system graph for global processes, or for those that interact with one, communication is required. This communication provides little impact on the network, unlike the earlier schemes which make very heavy demands in order to determine the true network status. Processes which are local at a particular

---

2: See

Bernstein, P.A., et al., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)", IEEE Trans. on Software Engg., SE-4 (3), May 1978, pp. 154-168.

Stonebraker, M.R., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", Memorandum No. UCB/ERL M78/24, Electronics Research Lab., Univ. of California, Berkeley, Calif., U.S.A., May 1978.

instant could become global at a later time necessitating the transmission of a collection of accumulated resource allocations to the various installations. Transitions of local processes to global status leads to a small incremental change in the size of the global system graph. The ensuing communication is still modest. Communication activity in our approach is modest in the sense that it is incremental and is dispersed over a period of time. Other approaches rely on simultaneous exchange of status for each site in the network. The transmission delays due to huge message traffic can lead to synchronization problems, which our more responsive method minimizes. Our approach also avoids the message congestion due to simultaneous transfer of large tables from every installation. The advantage of this method lies in its utility for on-line deadlock detection in distributed DBMS, which in turn means the corrective action can be taken earlier.

Given the system graph G_s, the two significant steps in the detection of deadlocks are:

(i) to determine the reachable sets of all the nodes in the system graph G_s, and

(ii) to find out if the necessary and sufficient conditions for the existence of a deadlock is fulfilled, by utilizing the reachable sets.

ALGORITHM B

Input: System graph (G_s).

Output: Processes involved in a deadlock(if detected).

B1: [Determine reachable sets of all nodes in G_s] For every node $N_j \in N$, determine the reachable set $R(N_j)$. That is, $R(N_j)$ is the set of all nodes in N such that there exists a directed path from N_j to these nodes.

B2: [Detect deadlock] If $N_j \in N$ is a process node such that $N_j \in R(N_j)$, then N_j is deadlocked. For every subset of process nodes, $\{Nj_i\}_{1 \leq i \leq k}$ such that $Nj_i \in R(Nj_i)$ for all i, $1 \leq i \leq k$, and $R(Nj_1) = R(Nj_2) = \ldots = R(Nj_k)$, the corresponding subset of processes is deadlocked.

Remarks: The proof of correctness of the algorithm is straight forward, and follows directly from Definition 2, Theorem 1, and Corollary 1.

We illustrate our approach by an example. Assuming that P_8 requests access to D_8 in the configuration of Figure 3.1, a PRW edge (P_8, D_8) is introduced. The inclusion of this edge causes P_8, P_5, P_7 and P_6 to be deadlocked. To detect such a deadlock, our mechanism determines reachable sets of all nodes in G_s. In the example considered, the reachable sets of these processes are the same and equal $\{D_8, P_5, D_9, P_7, D_7, P_6, D_{10}, P_8\}$. The existence of deadlock is detected by noting that each process in $\{P_5, P_6, P_7, P_8\}$ belongs to its own reachable set.

3.5 'On-Line' Detection

When dealing with concurrent database accesses, little is known about the probability of interference or deadlock.

For transaction processing systems, Peebles and Manning [1978] firmly believe that interference is rare, and that elaborate avoidance or prevention mechanisms would not be economical. We agree and advocate the use of deadlock detection in distributed systems. Further, to quote Le Lann [1978], "our conclusion will be that for systems which include a partitioned database and which provide for storage of pending requests, maintenance of internal integrity boils down to a problem of deadlock avoidance or detection with distributed control". As a consequence, and in view of the present day trend towards increased concurrent access in systems, we recommend the use of on-line deadlock detection in distributed systems. It is our belief that such a method contributes substantially to increasing concurrency.

In our view, deadlock prevention schemes are not justifiable for use in distributed systems. Processes that are not known to be nonconflicting need extensive coordination and, in general, substantial communication among installations is necessary before process initiation. This affects system performance by lowering the degree of concurrency. The past use of prevention principles was acceptable because of low levels of concurrency in systems rather than any inherent superiority.

In on-line detection, every installation can determine whether or not allocating one of its data resources to a process residing on another computer will lead to deadlock. This is facilitated by the ready availability at each

installation of the system graph and the reachable sets, which are continually updated as edges are added or deleted. The data resource allocation decision is transmitted by the access controller at the installation concerned to all others in the network. Thus, maintaining and updating the system graph for global interactions, at each installation, requires a low level of continual communication.

The on-line detection of deadlocks need be considered only for the following complete set of process-resource interactions:

a) a new process enters the system;

b) a new data resource is accessed;

c) a process runs to completion and releases data resources held[3];

d) a process in the system requests access to a data resource held by another process; and

e) a data resource held by a process is preempted from it.

3: In order to retain the "strong consistency" result of Eswaran et al. [1976], which requires that the processes be "well-formed" and "two-phase", a process is required to be divided into growing and shrinking phases. The first unlock action marks the beginning of the shrinking phase, after which a process cannot issue a lock request on any entity in the database until the release of all entities held by the process.
The actual implementation of a two-phase protocol (as in SYSTEM R) is to release all data resources held, at the completion of the process. (Private Communication from J.N. Gray, IBM Research Laboratory, San Jose, Calif., U.S.A., February 1979.)

3.6 Resolution of Process-Resource Interactions

(a) A new process enters the system and/or
(b) A new data resource is accessed: New process and/or data resource entry into the system introduces the respective nodes into G_s. An RPA edge is added to G_s whenever a new data resource is accessed either by an entering process or by one in the system. The request by an entering process for a data resource in the system may not be granted, thus introducing a PRW edge. In either case, a deadlock-free system continues to be so.

Assertion 1: If the system in a network configuration is deadlock-free, a new process entry into the system does not lead to deadlock in a circular wait condition. ■

Assertion 2: If the system in a network configuration is deadlock-free, accessing a new data resource does not lead to deadlock in a circular wait condition. ■

(c) A process in the system runs to completion and releases all data resources held: The process node, and all released data resource nodes which have no waiting-access requests, are deleted from G_s. Released data resources with a single waiting-access request are allocated to the corresponding processes. However, an allocation decision for a released data resource with multiple waiting-access requests is done in the manner indicated in Example 3.1 and according to the condition to be derived in Lemma 2, to avoid a potential deadlock.

Assertion 3: If the system in a network configuration is deadlock-free, then neither releasing the data resources held by a completed process for which there are no requests, nor allocating the released data resources for which there is a single waiting-access request leads to deadlock in a circular wait condition. ■

Example 3.1: For the configuration of Figure 3.1, let us assume that P_2 requests access to D_1 held by P_1, thus introducing the PRW edge (P_2,D_1) in G_s. Let P_3, which holds D_3, D_4 and D_6, run to completion. D_6 has no requests and hence is deleted from the system graph. D_3 is being waited for by P_1 and is allocated to P_1, whereas D_4 has two waiting-access requests from P_1 and P_4. Assume that P_4 issued its request before P_1 did. If the allocation of D_4 to P_4 is done in a FIFO manner, then processes P_1, P_2, and P_4 will be deadlocked. It is obviously more advantageous to make the allocation of D_4 to P_1 and let P_1 proceed, than to make the allocation of D_4 to P_4 and be deadlocked. This is crucial, especially when rollback and recovery in a network environment is expensive. Therefore, in Lemma 2, we give a necessary and sufficient condition to recognize such a situation and to avoid deadlock accordingly. Corollary 3 to Lemma 2 states that in the case of a deadlock-free system with multiple processes waiting for access to a released data resource, there exists at least one process such that an allocation made to this process maintains the system deadlock-free. In the case of multiple processes waiting in

a FIFO manner for access to a released data resource, the allocation is made to the first process which maintains the system deadlock-free. Further improvement may be possible by allocating the resource to the first process which not only maintains the system deadlock-free, but also has minimum waiting-access requests on other data resources.

Lemma 2: Let P_i and P_j be any two processes in a deadlock-free system with waiting-access requests to the data resource D_k. The allocation of the data resource D_k to P_i causes deadlock in a circular wait condition if and only if $P_j \in R(P_i)$ before the allocation

Proof:

(<==) (P_i,D_k) and (P_j,D_k) are PRW edges in G_s and since $P_j \in R(P_i)$, it follows that $P_j \in R(P_i) \supset R(P_j) \supset \{D_k\}$ and there exists a path from P_i to P_j without using the edge (P_i,D_k). Hence, the deletion of the edge (P_i,D_k) and the introduction of the edge (D_k,P_i) in G_s to allocate D_k to P_i, leads to a path from P_j to P_i through the edges (P_j,D_k), and (D_k,P_i), Therefore, $P_i \in R(P_j)$ ==> $R(P_j) \supset R(P_i)$. But $R(P_i) \supset R(P_j)$ ==> $R(P_i) = R(P_j)$. Thus, $P_i,P_j \in R(P_i) = R(P_j)$.

(==>) After D_k is allocated to P_i the processes are deadlocked, implying that $P_i,P_j \in R(P_i) = R(P_j)$. Now, we claim that $P_j \in R(P_i)$ before the allocation of D_k to P_i. Let us assume the contrary. That is, if $P_j \notin R(P_i)$ before the allocation of D_k to P_i, then the allocation of D_k to P_i should have added P_j to $R(P_i)$ which is a

contradiction, implying that $P_j \in R(P_i)$ before the allocation. ■

Corollary 3: Let $\{P_i\}_{1\leq i\leq n}$ be the processes in a deadlock-free system with waiting-access requests to the data resource D_k. There exists at least one process P_s $(1\leq s\leq n)$ such that the reachable set of P_s does not contain any process P_i for all i=1,2,...,n.

Proof: Assume the contrary. That is, there does not exist a process P_s such that $P_i \notin R(P_s)$ for all i=1,2,...,n. This in turn implies that for any process P_s $(1\leq s\leq n)$ there exists at least one process P_j such that $P_j \in R(P_s)$, where $j \in \{1,2,...,n\}$.

$$\Longrightarrow \quad R(P_s) \supset R(P_j).$$

Intuitively, there exists no process in the set $\{P_i\}_{1\leq i\leq n}$, whose reachable set does not contain that of another process in the set. Since there is a finite number of processes, $P_s \in R(P_j) \supset R(P_s)$. $\Longrightarrow P_j, P_s \in R(P_j) = R(P_s)$.
By Theorem 1, this is a deadlock situation, which is a contradiction. ■

Remarks: The processes in the set $S = \{P_i\}_{1\leq i\leq n}$ waiting for a resource D_k in a deadlock-free system can be topologically sorted[4]. The allocation of D_k to any of the processes which do not precede any other process in such a topological sort

---

4: For topological sorting refer: Knuth, D.E., Fundamental Algorithms, The Art of Computer Programming, 1, Addison Wesley Publishing Co., Reading, Mass., 1972, pp. 258-265.

of the set S, retains the system deadlock-free. Typically, these processes to which the allocation can be made, have minimum cardinality reachable sets. However, it is possible to find a process in S which does not precede any other process in the topological sort of S, but which has non-minimal cardinality reachable set by virtue of its waiting for processes other than those in the set S (since we allow more than one outstanding request for a process).

The situation in Example 3.1 arises because process P_4 has waiting-access requests for both D_2 and D_4. In this case our scheme detects a potential deadlock and avoids it accordingly, by virtue of Lemma 2. The potential for a query to be waiting for access to two data resources is illustrated in the Appendix B. It is unrealistic to restrict a process to have only one outstanding request, yet this has been the case in approaches by earlier authors - including the one by Goldman. Thus, our approach combines detection and avoidance principles in deadlock handling, and deals with multiple waiting requests in a realistic way.

Although, in operating systems, a process was allowed to have at a time more than one outstanding request; however, if the system was unable to satisfy all the outstanding requests at once, the process was required to drop them [Holt, 1972]. This in turn ruled out the occurrence of a situation analogous to case (c) developed in our approach. Consequently, researchers of the deadlock problem in databases did not allow more than one outstanding

request, thus avoiding the situation in which an allocation of a released resource (following the completion of a process) lead to a deadlock. On account of the fact that the requests in databases are data-driven and content-based the possibility of multiple outstanding requests is high. Thus, the results and the method suggested in this chapter to handle case (c) are new and original. Also, the combination of both the detection and avoidance principles to provide a mixed solution is the first of its kind in database systems.

(d) A process in the system requests access to a data resource held by another process: Since the request cannot be granted, the introduction of the PRW edge can lead to a cycle in G_s, meaning a deadlock. In this case, the reachable sets are updated appropriately, and a test for deadlock is carried out.

(e) Preemption of data resources held by a process: Preempting data resources is done when a process is aborted in an attempt to break a system deadlock. In such a case, the waiting-access requests of the aborted process are dropped, and the data resources held by the process are released.

Typical criteria for the selection of process(es) to be aborted are outlined below. However, the algorithms for selecting the process are non-trivial, and are not dealt with here.

(i) Abort a process which holds minimum number of data resources for exclusive access (preferably none), since this can result in reduced rollback costs.

(ii) Of all the deadlocked processes, abort the one which has used minimum CPU time.

(iii) Abort a process which involves rollback at a single installation, in preference to termination of one that leads to global rollback and consequent communication overhead.

(iv) Abort a process which has modified as few data resources as possible, and has interacted with other processes as little as possible, to minimize the cost of rollback.

Also, different analytic models, strategies, and approaches to rollback and recovery appear in [Chandy and Ramamoorthy, 1972; Chandy et al., 1975; Maryanski and Fisher, 1977].

3.7 "On-line" Deadlock Detection Algorithms

Bayer[1975,1976] has presented and analyzed an on-line transitive closure algorithm for deadlock discovery in databases. To our knowledge, on-line deadlock detection algorithms for distributed DBMS have not yet been proposed. In this section, we present procedures for updating reachable sets as interactions go on in the system. Further, these procedures are used in developing an on-line deadlock detection technique.

The step of prime importance in the on-line detection

of deadlocks is updating the reachable sets of the nodes in G_s. Maintaining G_s is fairly trivial, but calculating reachable sets of all nodes starting from scratch, every time an edge is added or deleted can be comparatively expensive. It is sufficient to design a good on-line algorithm for updating reachable sets, since the reachable sets need to be modified partly as edges are added or deleted from G_s.

Let N be the set of nodes of G_s. $R(N_i)$ is the reachable set of node $N_i \in N$ for all i, $1 \leq i \leq |N|$. Algorithm A, presented below, updates the reachable sets of G_s, given that a new edge (N_x,N_y) is added to the system. In step A1 of the algorithm, the reachable set of the node N_x is updated (if N_x existed in G_s) or created (if N_x is an entering node). In step A2 of the algorithm, we update the reachable sets of all those nodes, whose reachable sets originally contained N_x. Step A2 is never executed if N_x is an entering node, since no reachable set need be updated. Step A3 of the algorithm updates the system graph G_s.

ALGORITHM A

Input: (N_x,N_y), the new edge added to the system graph $G_s = (N,E)$.

Output: Updated reachable sets of all nodes in G_s and updated G_s.

A1: [Update reachable set of N_x]

1 if $N_x \notin N$ then $R(N_x) \leftarrow \phi$;

```
2   if N_y ∉ N then R(N_y) <- φ;
3   R(N_x) <- R(N_x) ⊔ R(N_y) ⊔ {N_y};
```

A2: [Update reachable sets of all nodes]

```
1   if N_x ∈ N then
2      begin
3        for i=1 until |N| do
4           if N_x ∈ R(N_i) then R(N_i) <- R(N_i) ⊔ R(N_x);
5      end;
```

A3: [Update the system graph G_s]

```
1   if N_x ∉ N then N <- N ⊔ {N_x};
2   if N_y ∉ N then N <- N ⊔ {N_y};
3   E <- E ⊔ {(N_x,N_y)};
```

Although, it is uncomplicated to update the reachable sets, when an arbitrary edge is added to G_s, it seems almost impossible to do so when an arbitrary edge is deleted from G_s. No better method than recalculating reachable sets seems feasible. On close examination, however, it becomes apparent that the only times when edges are deleted from G_s are:

(i) a process runs to completion and releases all data resources held; and

(ii) a deadlock is discovered, and at least one of the processes involved should be aborted, implying that all data resources held by aborted process(es) must be released, and also all access requests from the process (es) are to be dropped.

In case (i) the process is obviously not blocked and hence the corresponding process node in G_s is a sink. Thus, the edges dropped are only those that are directed to a sink. This is a very simple case, and hence an algorithm can be devised to update the reachable sets. Whereas, for the deadlock situation of case (ii), there exists no sink in G_s. Therefore, aborting a process and rolling it back, requires us to recalculate the reachable sets. Maintenance of these sets by incremental updates considerably decreases the chances of synchronization error and that of the problem of the system graph becoming obsolete in all interactions (a) to (d) discussed in Section 3.6. However, for interaction (e) complete reconstruction of reachable sets is necessary.

We present ALGORITHM T to update the reachable sets of all nodes given that an edge (N_w,N_z) is deleted, where N_z is a sink in G_s. In updating G_s, the edge (N_w,N_z) is deleted from E, but the appropriate node deletions are done elsewhere (in the algorithm that is to be proposed for on-line detection of deadlocks).

ALGORITHM T

Input: Edge (N_w,N_z), deleted from the system graph, where N_z is a sink.

Output: Updated reachable sets of all nodes in G_s, and updated E of G_s.

T1: [Update reachable sets of all nodes]

```
1       for i=1 until |N| do
```

2 $\underline{\text{if}}\ N_z \in R(N_i)\ \underline{\text{then}}\ R(N_i) \leftarrow R(N_i) - \{N_z\};$

T2: [Update the set of edges E of G_s]

1 $E \leftarrow E - \{(N_w, N_z)\};$

We, now propose ALGORITHM S to detect deadlocks on-line, for all cases discussed in Section 3.6. ALGORITHMS A and T are extensively used in ALGORITHM S, to update reachable sets as edges are added to or deleted from G_s. Step S1 of the algorithm updates reachable sets for a new process and/or data resource entry. Step S2 deals with the case in which a process runs to completion and releases all data resources held. In S2a, the reachable sets are updated by deleting edges, corresponding to the release of data resources. Step S2b updates the set of nodes in the system graph for the released data resources without any waiting-access requests. Allocation of released data resources with single waiting-access requests is done in S2c. In step S2d, for a set of processes $\{P_w\}$ waiting for a released data resource, the condition in Lemma 2 is tested successively between pairs of processes until a process is found[5] whose reachable set contains no other process in $\{P_w\}$. The data resource should be allocated to such a process to avoid a potential deadlock. In step S3, the case when a process requests access to a data resource held by another process is dealt with. Step S4 deals with the case when a process

---

5: An alternative method is to allocate the resource to a process with minimal reachable set in $\{P_w\}$ as discussed in the remarks following Corollary 3.

is aborted to break a deadlock. In step S5 we carry out a test for the existence of deadlock, for the cases dealt with in steps S3, and S4.

ALGORITHM S

S1: [A process enters the system, or a new data resource is accessed] As a consequence of a new process entry, or accessing a new data resource, or both, let the new edge added be (N_i,N_j). Apply ALGORITHM A with the edge (N_i,N_j) as input, and STOP.

S2: [A process in the system runs to completion and releases all data resources held]

S2a: [Update reachable sets by deleting each edge] Let the process which runs to completion be P_k, and the data resources released be $Dj_1,Dj_2,\ldots,Dj_s$. Apply ALGORITHM T with the edge (Dj_i,P_k) as input, for all i, $1\leq i\leq s$. Delete P_k from the set of nodes, N of G_s.

S2b: [Update G_s by deleting, from N, released data resource nodes without any waiting-access requests] If there is no edge directed to Dj_i then delete Dj_i from N for all i, $1\leq i\leq s$.

S2c: [Allocate released data resources with a single waiting-access request] For every released data resource Dj_x for some $x \in \{1,2,\ldots,s\}$, with single waiting-access request from process P_y (say). Apply ALGORITHM T with (P_y,Dj_x) as input, and apply ALGORITHM A with (Dj_x,P_y) as input.

S2d: [Allocate released data resources with multiple waiting-access requests] For every released data resource Dj_x, for some $x \in \{1,2,\ldots,s\}$, with multiple waiting-access requests from a set of processes $P = \{Py_1, Py_2, \ldots, Py_m\}$, find a process Py_k such that $Py_i \notin R(Py_k)$ for all i, $1 \leq i \leq m$. Apply ALGORITHM T with (Py_k, Dj_x) as input[6]. Apply ALGORITHM A with (Py_i, Dj_x) as input for all i, such that $i \neq k$ and $1 \leq i \leq m$. Apply ALGORITHM A with (Dj_x, Py_k) as input. STOP.

S3: [A process in the system requests access to a data resource held by another process] Let the process be P_i, and the data resource be D_j. Apply ALGORITHM A with the edge (P_i, D_j) as input. GO TO STEP S5.

S4: [A data resource held by a process is preempted from it, to break a deadlock] Let the aborted process be P_j. Delete from the set of edges, E of G_s all edges directed to P_j, and all edges directed from P_j. Apply Step B1 of ALGORITHM B with G_s as input, and GO TO STEP S5.

S5: [Detect deadlock (if any)] Apply step B2 of ALGORITHM B to detect deadlocks, (if any). STOP.

For the purpose of illustration of the on-line deadlock

6: This has the effect of deleting Dj_x from the reachable sets of all Py_i for $1 \leq i \leq m$. Thus the reintroduction of the edges (Py_i, Dj_x) for all i, such that $i \neq k$, and $1 \leq i \leq m$, is necessary.

detection scheme, we utilize the network configuration of Figure 3.1. We assume the introduction of a PRW edge from P_2 to D_1, consequent to P_2 requesting access to D_1, and an RPA edge from D_7 to P_5 as a result of P_5 requesting shared access to D_7 which is also held for shared access by P_6.

Illustration of ALGORITHM S

S1: [A process enters the system, or a new data resource is accessed or both] Let the new edge added be: $(P_\emptyset, D_1)$ (new process entry), or $(D_\emptyset, P_2)$ (new data resource entry), or $(D_\emptyset, P_\emptyset)$ (both). Apply ALGORITHM A with $(P_\emptyset, D_1)$, or $(D_\emptyset, P_2)$, or $(D_\emptyset, P_\emptyset)$ as input. STOP.

S2: [A process in the system runs to completion and releases all data resources held] Let the process which runs to completion be P_3, and the data resources released be D_6, D_3, and D_4.

S2a: [Update reachable sets by deleting each edge] Apply ALGORITHM T with (D_6, P_3) as input; Apply ALGORITHM T with (D_3, P_3) as input; Apply ALGORITHM T with (D_4, P_3) as input; Delete P_3 from G_s.

S2b: [Update G_s by deleting released data resource nodes without any requests] Delete D_6 from G_s.

S2c: [Allocate released data resources with a single waiting-access request] Apply ALGORITHM T with (P_1, D_3) as input; Apply ALGORITHM A with (D_3, P_1) as input;

S2d: [Allocate released data resources with multiple waiting-access requests] For the released data resource D_4 the condition in Lemma 2 dictates its

allocation to P_1, rather than to P_4. Apply ALGORITHM T with (P_1,D_4) as input[7]. Apply ALGORITHM A with (P_4,D_4) as input; Apply ALGORITHM A with (D_4,P_1) as input. STOP.

S3: [A process in the system requests access to a data resource held by another process] Let the process be P_8, and the data resource be D_8. Apply ALGORITHM A with (P_8,D_8) as input. GO TO STEP S5.

S4: [A data resource held by a process is preempted from it, to break a deadlock] Let the aborted process be P_5. Delete from G_s the edges (D_8,P_5), (D_7,P_5), and (P_5,D_9). Calculate the reachable sets of all nodes of G_s. GO TO STEP S5.

S5: [Detect deadlock (if any)] Apply step B2 of ALGORITHM B, to detect deadlocks, (if any). STOP.

3.8 Discussion

3.8.1 Communication Requirements

In distributed database performance, the communication time is a critical factor to be optimized. Consequently, inter-installation communication in distributed DBMS can result in conspicuous performance degradation. Thus, the impact of inter-computer communication upon system

---

7: This has the effect of deleting D_4 from the reachable set of P_4, and those of the nodes from which D_4 was access ible. Thus, the reintroduction of the edge (P_4,D_4) is necessary.

performance should be an important consideration in the treatment of deadlocks in distributed systems. Not only is an efficient network communication mechanism essential, but also an effective deadlock handling technique, with minimal communication requirements.

In the approach proposed in this chapter to detect deadlocks, the communication needs are quite modest. Every time a data resource is allocated to a process, a process releases a data resource, or a process is made to wait for a data resource, the access controller at the installation in which the data resource resides, sends information to that fact to all other installations in the network. Such communication is necessary only for processes and data resources which are global in nature, as discussed in Sections 3.4 and 3.5. This facilitates maintaining the system graph at each installation. Thus, maintaining and updating the reachable sets is done at each installation to detect deadlock, without the necessity of transmitting huge tables among installations.

In the approaches by Chu and Ohlmacher[1974], and Maryanski[1977], process sets and shared data lists are respectively, transmitted over communication channels to each installation, resulting in huge message traffic. Chandra et al. [1974] transmit resource tables, which contain information pertaining to processes allocated local resources, processes waiting for access to local resources, local processes allocated remote resources, and local

processes waiting for access to remote resources. Similarly, Mahmoud and Riordon[1977], in their distributed approach, in a network of 'n' computers require the transmission of (n-1) identical messages containing status and queues of files. Also each installation receives (n-1) different messages from other installations. In other words, all these approaches basically require the transmission of large tables among installations, resulting in tremendous communication. Goldman[1977], in his approach requires the creation of OBPL, which is expanded at each installation, and transmitted from there to the next. The expansion and transmission of OBPL's is done till a decision is arrived at, whether or not a deadlock exists. Even though, Goldman's approach is better than those of other authors discussed above, the OBPLs go through several installations sequentially or several times between the same two installations till a deadlock is detected. Thus, the approach could result in undue waiting period, during which a totally new network situation could arise. All these approaches suffer from the great drawback that the size of the tables to be transmitted increases enormously as the unit of identifiable resource decreases in size (e.g., from files to records).

8.3.2 Responsibilities of Data Base Administrator

The complexity of the function of a data base administrator (DBA) increases enormously with the distribution of DBMS over a network of computers. In a network environment, the

actions of the operating system which manages application jobs (processes), and that of the DBA who maintains the process integrity and the consistency of the database have to be coordinated. A thorough understanding of the relationships among concurrency controls, processors, processes, deadlock handling and recovery techniques, communication aspects and protocols, etc., is necessary for the DBA as well as the operating system. The relevance of such coordination may necessitate the importance of having a higher authority over both the DBA and the operating system. This area probably requires a deeper study, especially in a network environment.

As more and more data is integrated over a network of computers, the database becomes more accessible to a larger number of diverse application jobs, thus contributing to generality and flexibility. Simultaneously, several operations like detection of deadlocks, recovery from deadlocks, communication requirements, etc., become extremely complex, demanding improved operational efficiency. This boils down to a classical trade-off between generality and efficiency, of very common occurrence in commercial data processing. It is in balancing these two apparently conflicting factors that the role of DBA assumes great importance. Very significantly, the decisions made by DBA will have substantial impact in a system on a network of computers should an on-line deadlock detection technique like the one proposed in this chapter be implemented.

3.9 Highlights of our Proposal

It is difficult to estimate the performance effects of deadlock detection or deadlock prevention in distributed DBMS, since communication time is critical. Because of the complexity of distributed DBMS a significant factor in handling deadlocks would be operational efficiency. But the communication aspects make it impractical to estimate the performance of such algorithms analytically. Once distributed DBMSs become a common reality experimental data can be gathered to measure the performance. Nevertheless, our proposal has the several advantages shown below:

* In the deadlock detection approach proposed in this chapter, the communication needs are quite modest in the sense that it is incremental and dispersed over a period of time, as outlined in Sections 3.4, 3.5 and 3.8.1.
* The technique of deadlock detection suggested in this chapter identifies the processes directly responsible for the deadlock (Corollary 1). In general, it is possible to find a process blocked forever but not deadlocked, by virtue of its waiting for a process which is involved in a deadlock (Corollary 2).
* Processes are never delayed by our technique, because a process whose request for a data resource can be granted may proceed without waiting for the deadlock detection mechanism. The DBA can design a scheme to invoke the detection mechanism for every 'X' units of time, or for every 'Y' accesses granted, or for every 'Z' accesses

not granted, or any combination of these.

* In our approach, a process may have any number of outstanding requests simultaneously. For most others [e.g., King and Collmeyer, 1973; Goldman, 1977], a process is restricted to have at most one outstanding request. In real world applications this restriction is not practical, as demonstrated in Appendix B. Thus our approach is more general and realistic.

* When a number of readers share a data resource, our algorithm does not require any special treatment, unlike in Goldman's approach where one different copy of the OBPL is formed for each such reader. In the case of shared access his approach leads to a heavy overhead in computation and in communication.

* Our proposal deals with every request in the same manner, and can be considered a unified approach, since the detection technique does not classify requests according to the relationship between the origin of the process and the installation of residence of the data resource accessed. In all other approaches discussed in Section 2.6.3, the algorithms deal with each access request according to the classification of the request.

CHAPTER 4

SYSTEM RECOVERY IN DISTRIBUTED DATABASES

4.1 Introduction

For distributed databases, very interesting concurrency controls have independently been designed [Thomas, 1977; Bernstein et al., 1978; Rosenkrantz et al., 1978; Stonebraker, 1978]. In particular, Bernstein et al. propose an effective method to maintain mutual consistency of multiple copies of databases and internal consistency of each copy of the database. This method incorporates deadlock prevention principles. Excellent response time is guaranteed for all transactions that do not conflict. Transactions that cannot be shown to be nonconflicting need extensive coordination and, in general, substantial communication among installations is required, as a part of the deadlock prevention mechanism.

Knowledge about the probability of interference or deadlock in concurrent database accesses is relatively unknown. Peebles and Manning[1978] strongly believe that interference is rare in transaction processing systems, and do not support the idea of elaborate avoidance or prevention mechanisms which are considered uneconomical. We agree and

have strongly advocated in Chapters 2 and 3 the use of deadlock detection in distributed systems. Further, to quote Le Lann[1978], "our conclusion will be that for systems which include a partitioned database and which provide for the storage of pending requests, maintenance of internal integrity boils down to a problem of deadlock avoidance or detection with distributed control". In view of Peebles and Manning's belief and Le Lann's conclusion, we feel that our on-line detection technique provides adequate measures to protect the database against process failures. However, neither our scheme nor that of Bernstein et al. are robust enough for system crashes.

Users of shared databases presume that the consistency and correctness of information upon which they work is preserved, under a wide variety of system malfunctions. In a network environment the problems faced in maintaining data accuracy are even more severe. In this chapter we present a robust approach for system recovery from crashes.

4.2 The Problem, Environment, and Basic Strategy

A system crash normally requires database reconstruction by for instance, reloading a previous save and repeating the updates on the database from that checkpoint through use of an audit trail. In a batch environment such a procedure may be expensive, inconvenient and time-consuming. On the other hand, in real-time transaction processing systems, failures can have more disastrous

effects. For instance, after a successful reloading from a previous checkpoint, it is unreasonable to expect that users will remember long sequences of transactions. Even if they did remember, expecting them to perform their own recovery is not realistic. In some systems, continued processing after a partial recovery may not be a serious problem. For example, a supermarket inventory control system might as a consequence attempt to oversell a product, probably causing little more than some inconvenience and minor embarrassment. On the other hand, if we are dealing with airplane seats, bank withdrawals or paychecks, a partial recovery could have severe repercussions.

In essence, we are considering the problem of reliable operation of a distributed data management system in the presence of failures. More formally the problem is: Given:

--an arbitrary computer network with distributed control, and an appropriate communication system, and

--an integrated database, appropriately partitioned and/or replicated, is distributed over the network.

Design a reliable method of system recovery that maintains database consistency during update transactions in the presence of either system failure or communication breakdown.

Properties that characterize a "good" solution are:

--Simplicity: The techniques used for detecting a failure and for recovery must be simple, and should not incorporate any elaborate methods.

--Tolerable overhead: Under normal circumstances a high overhead results in fast recovery, while lower overhead provides better performance but slow recovery. We stress the importance of round-the-clock access to the on-line system, and advocate that the overhead involved in maintaining recovery data should be directly proportional to the value or sensitivity of the data accessed.

--Consistency: Mutual and internal consistency of the database should be maintained after recovery. In the event of brief communication failures, the design of an effective procedure is necessary for reconstructing a consistent database from two or more isolated fragments. A complete technical solution to this problem in a partitioned network is not in sight.

--Partial operability: The system should continue to operate in the case of failures at one or more installations.

--Avoiding global rollback: As far as possible, rolling back all the executing transactions to some common checkpoint in time should be avoided.

--Reliable communication: Guaranteed delivery of messages is a necessary requirement to ensure reliable recovery and maintenance of consistency.

Environment:

The environment with respect to transactions, communication aspects, failure and recovery, the timestamping mechanism, and the applicability to different types of networks is explained in detail:

1. Transaction categories: The traffic processed by a distributed database system is assumed to fall in one of the three broad categories. Special purpose transactions of the type found in airline reservation, medical information, or banking systems require extra care because of the sensitive nature of the application. Less sensitive applications which involve dedicated or self-contained equipment, are categorized as local transaction systems, (e.g. warehouse inventory control, payroll production, or student record systems). Finally the third category encompasses global transactions which access data stored in two or more different installations via a communication line.

2. Communication aspects: Every message sent from an installation is assumed to reach the destination eventually. Messages which arrive at a destination computer from a common source are processed in the order of initiation. However, the messages from two different installations to a particular installation may be processed in their arrival order. To ensure that any message sent from one installation to another is eventually delivered, despite a finite possibility of failure of both the source and the destination, the concept of message spoolers [Hammer and Shipman, 1978] is employed. For instance, if an installation N_1 wants to communicate a message to crashed installation N_2, N_1 establishes several copies of its message at different

installations with the help of message spoolers. Upon recovery, N_2 picks up these messages sent to it while it was down, regardless of whether N_1 has crashed since it sent the message.

3. Failure and recovery: The failure of an installation is assumed to have been detected by other installations by the time they need to know of its crash. Upon the recovery of that installation, the other installations are assumed to be informed promptly. Appropriate recovery procedures, to be proposed in this chapter, are assumed to be in force as soon as the crashed system comes up.

4. Timestamping: A unique timestamp generator is supposedly in effect for every data modification, for every aspect of recovery data stored, and for every message transferred. The successively generated timestamps have monotonically increasing value. Timestamps are used for synchronization.

5. Networks: The approach is equally applicable to all types of networks from "store-and-forward" communication (e.g., ARPANET [McQuillan and Walden, 1977]) on one side of the spectrum, to "broadcast" networks (e.g., ETHERNET [Metcalfe and Boggs, 1976]) on the other.

The basic strategy:

Although jobs consist of retrieval and update

transactions, these two groups may be subclassified and recovery protocols defined which take advantage of known properties of each transaction class.

4.3 Transaction Classification

Transactions[1] are classified according to the degree of requirement for fast system recovery. The pre-definition of transaction classes can be performed by the Data Base Administrator during database design, depending on either the information about the specialty of the data stored (e.g., airline reservation system) or by gathering statistics about the database usage (e.g., predominantly local transactions). The defined classes can then be appropriately matched with recovery protocols necessary to maintain the system specification. The pre-definition of transaction classes does not constrain the system from accepting any transaction, but rather facilitates the use of more efficient and cost effective recovery schemes for transactions of known or predictable behaviour.

4.3.1 Retrieval Transactions

We classify retrieval transactions into two kinds. The transactions that do not require read locks will be referred to as that of class T1. Transactions of class T1 are those that do not really care if someone else is modifying the

---

1: "Processes" and "transactions" are used interchangeably.

same data. For example, in an automated library database, where the majority of users are those who search for the presence of a particular document, one is not disturbed by a transaction which at the same time changes the status of this document from "available for loan" to "loaned to Mr. Smith". Such a user should be allowed to access the database for reading even if the data is locked for modification. This not only enhances concurrency in the database, but also helps us to devise a simple and inexpensive recovery protocol for a large number of retrieval transactions of this kind. System performance is enhanced on account of the reduction in locking costs.

Transactions of class T2, on the other hand, are those that require locking of the shared data for reading, and include all those that are not in T1. A transaction of class T2 has to wait for access for any data that is locked for modification, and also any data locked for reading by a transaction of class T2 cannot simultaneously be locked for modification by any other transaction.

4.3.2 Update Transactions

(a) Special purpose systems, which basically serve one specialized application but have lived up to expectations as a distributed database system, are considered here. Each specialized system has its own recovery requirements.

In the case of information systems for which the

probability of failure is required to be small, such as those needed for space shuttle applications, aircraft flight control, and to some extent airline reservation systems, speedy system recovery and data validation is essential. Only very occasional small losses can be tolerated. Thus, the recovery data and protocol are quite elaborate. We shall refer to this class of transactions by U1. For the space shuttle and airplane traffic control applications, standby systems provide further protection.

A class of specialized systems includes those where the requirement for user-definition of user-owned information is high, and yet data sharing is very important, and the data files are owned locally by the creator of that data, but are accessible to others through a network. For instance, in a distributed medical information system, the producers of information are geographically separated, like physician's offices, pharmacies, laboratories and hospitals. Banking and telephone systems can also be classified according to this kind of transaction, which we shall refer to by U2. For computerized telephone systems, very infrequent isolated small breakdowns can be tolerated, likewise in banking systems, significant processing and storage facilities are typically incorporated into computer terminals. Such terminals provide data input and possibly limited data validation, even when the main

computer system is down. Moreover, the class of transactions U2 do not require recovery techniques as elaborate as for the class U1.

(b) Systems with predominantly local transactions, for instance a university student database, a payroll system, or an inventory control system for a chain of supermarkets, are classified as U3. Commonly, over 95-99% of the transactions are local [Stonebraker, 1978]. The transactions of class U3 can put up with system crashes better than those of classes U1 and U2, and hence, fairly inexpensive recovery protocols and recovery data maintenance are allowable.

(c) U4 is the class of transactions which access data stored (not redundantly) in two or more installations. This class may include Personnel Management and Budgetary Accounting systems. Typically, a transaction of this class accesses data stored in the installation where it originates, and travels to another for further processing and so on.

(d) The class of transactions U5 is that which accesses data stored (redundantly) in two or more installations, handling fairly generalized transactions. An appropriate update algorithm [Bernstein et al., 1978] that maintains internal and mutual consistency of the database is assumed. A transaction of this class accesses the data stored in the installation where it originates, and

travels to the nearest installations where the redundant data is supposedly stored for further processing and so on. In practice the U4 class of transactions may be thought of as a subset of U5.

(e) U6 includes the transactions that were not anticipated ahead of time. On account of their unpredictable or unexpected behaviour, these transactions need extensive audit data and an elaborate protocol for system recovery. Typically transactions for new applications for which there is insufficient classification data are included in this class.

4.4 System Recovery Protocols

The approach outlined here differs from many other methods because it does not assume that every transaction requires recovery data (redundant data stored to make recovery possible) maintained in the form of "audit trails" (recording of 'who did what to whom, when, and in what sequence'). Instead, appropriate inexpensive system recovery protocols are proposed for transactions whose behaviour can be pre-determined.

If a transaction belongs to class T1, access is allowed to the data to be retrieved irrespective of whether the data is locked by any other transaction or not. The protocol in the case of a transaction of class T2 requires waiting if the data is locked for modification. The transaction is

free to proceed if the data is locked for shared use (reading). Neither class T1 nor class T2 transactions require maintenance of recovery data for crash resistance. However, a mechanism should be provided to indicate to the process:

(i) its wait state, if waiting for locked data; or

(ii) if the data requested resides in a site which has crashed or if all communication links to the site have failed.

Recovery protocols for update transactions:

A. Recovery Protocol SP1:

This is required by transactions of class U1, which may originate from remote locations and terminals.

1) The originating transaction is uniquely timestamped (i.e., two transactions starting at the same time from different sites are assigned distinct timestamps).
2) Provided the timestamp of the accessed data is less than that of the transaction, synchronize[2] the clocks of other sites to the timestamp of the transaction, provided it has to access data from these sites. Otherwise reassign a higher timestamp to the transaction, and repeat step 2.
3) Modified data is timestamped[3] at the site (say N_1). An audit trail is maintained by using a write ahead log

---

2: This step is included to provide synchronization of time clocks necessary for updating of replicated data in a distributed database.

3: Timestamp with the system-time at the moment of creation of the entry.

protocol, which requires that the audit trail be written to non-volatile storage before the database is updated [Gray, 1978].

4) If such modified data is redundantly stored (say at site N_2), then site N_1 sends a timestamped update message to be written to the audit trail at site N_2. Subsequently the database is updated at site N_2.

5) Periodic incremental dynamic dumping [Rosenkrantz, 1978] of the database is carried out to provide checkpoints.[4] Incremental dynamic dumping can be facilitated by maintaining a differential file [Severance and Lohman, 1976].

Remarks: The timestamps uniquely identify the transactions in the recovery data. An audit trail facilitates crash resistance, backing out of any transaction, and allows certification of system integrity, when necessary. Incremental dumping can be carried out frequently to provide recent checkpoints which, in conjunction with the audit trail, helps speedy system recovery. Maintaining a differential file as an "add set", "delete set", and "change set" makes dynamic dumping easier.

4: For this case a strategy for optimal checkpointing is suggested in this chapter.

B. Recovery Protocol SP2:

Transactions of class U2 require a less elaborate protocol than SP1. Incremental dumping here need not be either dynamic or frequent, relative to SP1.

1) Same as step 1 of SP1.
2) Same as step 2 of SP1.
3) Modified data is timestamped at the site (say N_1), and the recovery data is stored in the form of a differential file. A write ahead log protocol is used to first copy the differential file to non-volatile storage, before certifying process termination.
4) Sites which store the modified data redundantly are sent a timestamped update message. When a site receives such a message, the update communicated is written to the differential file maintained at that site.
5) At a pre-determined point in time the differential file is merged with the database to provide an up-to-date database, which is then dumped for use as a checkpoint.

C. Recovery Protocol SP3:

Class U3 transactions which are predominantly local in nature require dumping once every 24 hours or so. In order to provide crash resistance, recovery data is stored in the form of a differential file.

1) Timestamp the originating transaction.
2) Store recovery data in the form of a differential file maintained at that site using a write ahead log protocol.
3) Once every 24 hours (or as determined by the DBA) merge

the differential file with the database to update the data stored. Checkpoint the database after every such major update.

D. Recovery Protocol SP4:

Transactions of classes U4 and U5 use this protocol. An appropriate algorithm [Bernstein et al., 1978] for updating and maintaining consistency of the database in a redundant case is assumed.

1) Same as step 1 of SP1.
2) Same as step 2 of SP1.
3) Same as step 3 of SP2.
4) Same as step 4 of SP2.
5) Same as step 5 of SP2.

E. Recovery Protocol SP5:

The class U6 of unanticipated transactions, (whose behaviour with respect to updating data could not be predetermined) use the same protocol as SP4 except that

(i) an audit trail is maintained at step 3 of SP4 using write ahead log protocol, and

(ii) the update message communicated to another site in step 4 is first written to the audit trail, before committing the update.

4.5 Optimal Policy for Checkpointing

The optimization problem involves balancing the unavailable time during the creation of a checkpoint against

the unavailable time after a failure, when a saved version of the database is being updated from the audit trail. Several assumptions made in formulating this problem are specified below:

(1) For our convenience, a database formed from a relation in Codd's[1970] relational model[5] is used. The relation is assumed to have a time-varying number of tuples dependent on the update activity.

(2) For any given small interval of time, the number of transactions processed is proportional to the time interval. This still allows the constant of proportionality to differ for different intervals, but it is dependent on the traffic, which can be pre-determined. Thus, for a given time interval from $t_\emptyset$ to t_1, the number of transactions processed is assumed to be equal to $K_\emptyset * (t_1 - t_\emptyset)$ where $K_\emptyset$ is the transaction processing rate in the interval $(t_\emptyset, t_1)$, assumed constant.

(3) The average numbers of tuples read and written by a transaction is equal to r and w respectively. The quantity r is always greater than or equal to w since every tuple written has to be read prior to an update, but not vice versa.

(4) A system failure may occur at any time.

(5) The cost of checkpointing a relation is assumed to be proportional to the number of tuples in the relation.

5: The analysis is applicable to any model of data representation, despite the assumption about the relational model.

Let the unit cost per tuple be 'c'. Similarly, the cost of reloading a checkpointed database is assumed to be 'E' times the cost of checkpointing.

(6) The cost of maintaining an audit trail is dependent on the sum of the number of tuples read and the number of tuples written (the length of the entries made in the journal). The cost of using an audit trail to recover after a failure is 'F' times the number of entries for tuples modified.

Since the assumptions are independent of the size of the database we shall consider a single relation for our analysis. Let $t_\emptyset$, t_1, t_2, ..., t_k be points on a time scale such that the rates of processing transactions in the intervals $(t_\emptyset,t_1)$, (t_1,t_2), ..., (t_{k-1},t_k) be different constants, $\{K_i\}$, in each interval. Thus, the number of transactions processed during the time period from $t_\emptyset$ to t_k is:

$$\sum_{i=\emptyset}^{k-1} K_i * (t_{i+1} - t_i)$$

Therefore the numbers of tuples read and written during the time period $t_\emptyset$ to t_k is:

$$(r + w) * \sum_{i=\emptyset}^{k-1} K_i * (t_{i+1} - t_i)$$

Let us assume that a system failure occurs at time t_f, $(t_f > t_k)$.

The cost of reloading a saved version (from time $t_\emptyset$), had checkpointing not been done at t_k, is proportional to the cost of checkpointing at $t_\emptyset$ (C_c). (Assumption 5.)

$$C_c = E * c * N_\emptyset,$$

where E is the save/restore checkpoint ratio, and $N_\emptyset$ is the number of tuples checkpointed from the given relation at time $t_\emptyset$.

The reprocessing cost of the audit trail (C_{au}) for recovery is proportional to the number of tuples updated between $t_\emptyset$ and t_f. (Assumption 6.)

$$C_{au} = F * w * [(\sum_{i=\emptyset}^{k-1} K_i * (t_{i+1} - t_i)) + K_k * (t_f - t_k)]$$

where F = maintenance/processing audit trail ratio.

Total recovery cost (if checkpoint was not taken at t_k):

$$A = C_c + C_{au}$$

Total cost of recovery + cost of checkpointing at t_k (if the database was checkpointed at t_k):

$$B = E * c * N_k + F * w * K_k * (t_f - t_k) + c * N_k,$$

where N_k = the number of tuples checkpointed at time t_k.

If A > B, then it is worthwhile checkpointing at t_k. This yields the condition that,

$$F * w * \sum_{i=\emptyset}^{k-1} K_i * (t_{i+1} - t_i) \text{ be greater than}$$

$$C * [((E + 1) * N_k) - E * N_\emptyset)].$$

Intuitively, if the number of tuples added and deleted are approximately equal, and/or if the number of tuples in the relation or database is large compared to the number of tuples updated, we have N_k approximately equal to $N_\emptyset$. This in turn, leads the above condition to be interpreted as follows: Checkpointing at a time t_k is cost-effective, if the cost of processing the audit trail for recovery from the previous checkpoint $t_\emptyset$ to t_k exceeds the cost of checkpointing at time t_k.

Remarks: The optimal policy for checkpointing suggested for protocol SP1 here is derived using a very simple model. The assumption about the number of transactions processed in a given interval of time is realistic since for a small interval the time-dependent K_i's can be estimated. This is especially true in the environment we have considered for system recovery where prior transaction classification is done at database design. The policy determines the checkpoint dynamically as the transactions are processed, and is different from earlier fixed interval approaches. The feasibility of its implementation is enhanced by virtue of the parameters involved. All the necessary parameters can be pre-determined from either the information provided by DBA or from database usage statistics and expected behaviour of certain transactions. Consequently, the approach is new and practical.

4.5.1 Audit Trail Maintenance Policy

The decision to maintain audit trails should be dependent on some of the following considerations.

(i) Necessity of checking for security breaches;

(ii) Enforcing consistency requirements and authorized access to data;

(iii) Record on-line transactions for automatic recovery in the case of a system crash;

(iv) Backing out any single transaction, and recovering from deadlock.

Remarks: The audit trail should be physically reconstruct-

able if damage occurs to it. Damage repair to the audit trail is critical, which if not done invalidates the system recovery. Such recreation of the recovery data may be effected by either duplication of the audit trail, or use of Hamming codes and a salvager. Should related recorded data in the audit trail be distributed between physically separated audit trails or within the same audit trail, a mechanism for synchronization is necessary. Common transaction identification (the unique transaction timestamp) provides a criterion for synchronizing the distributed recovery data.

4.6 Domino Effect (Global Rollback)

Verhofstad[1977] describes a recovery scheme which is implemented for a filing system supporting a single user. In this mechanism, recovery and crash resistance is provided within the concept of a "recovery block" [Randell, 1975]. In the case of a system failure/crash inside the scope of a recovery block, the system will be "rolled back" to the state that existed upon entry to the recovery block. Checkpointing at the beginning of every recovery block is done dynamically. "Commitment" at the end of a scope, or "undoing" within or at the end of a scope can be achieved by invoking procedures. Verhofstad's mechanism also provides schemes to define a scope dynamically, and to back out on request, in case of a failure.

The question of recovery in systems with multiple

concurrent updates by simultaneously executing processes has been the topic of ongoing research. Progress has not been thorough or complete, because of several significant problems encountered among interacting processes. Verhofstad[1978] has stated the major difficulties faced by the designers of System R [Astarhan et al., 1976] for recovery in a multi-user environment. Consequently, the scheme is now supposedly used for recovery from total system failure only.

In a transaction processing system, the abnormal termination of a process has disastrous effects on the consistency of the database. In an interactive environment, the data modified by an abnormally terminating process may possibly have been used by others which in turn perform additional modifications to the database. This phenomenon of a process generating additional incorrect data can cascade throughout the database. Consequently, a large number of processes may be operating with potentially invalid or contaminated data. Therefore, the elimination of the effect on the database, by backing out an abnormally terminating process, is an essential part of any recovery technique. A major difficulty that is faced here is that the process being backed out may require the backing out of other processes creating what Randell calls a "domino effect".

The domino effect is illustrated with an example in Figure 4.1 by using three processes P_1, P_2, and P_3. Solid

lines directed from left to right show the progress of each of these processes with time. In this example each process, P_i, has entered four recovery blocks, but has not yet exited any of them. The times at which processes enter recovery blocks, referred to henceforth as recovery points are represented by R_{ij} for $1 \leq j \leq 4$, for every process P_i. The dotted lines between processes indicate process interactions. For instance, in Figure 4.1, interactions have taken place between processes P_1 and P_2 at times t_2, t_3, t_4, and t_8. Similarly, at times t_1, t_5, t_6, and t_7 between processes P_2 and P_3. Should process P_1 now fail at the current time T_p, it will have to be backed up to its most recent recovery point R_{14}. No other processes will be affected, since between R_{14} and T_p no interactions have taken place with process P_1. On the other hand, suppose that process P_2 fails at T_p, resulting in backing up P_2 to its newest recovery point R_{24}. Since R_{24} is prior to an interaction at time t_8 with process P_1, P_1 must be backed up to its first recovery point which precedes t_8, that is R_{13}. However, if process P_3 is to be backed up due to failure, all the processes P_1, P_2, and P_3 will have to be backed up right to the beginning of each process, thus illustrating the domino effect in extreme.

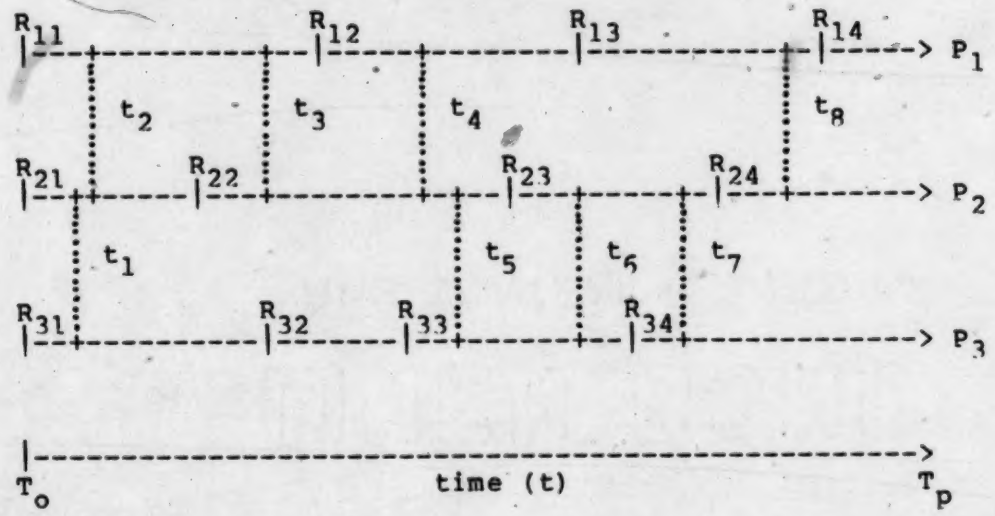


Figure 4.1: Domino Effect.

4.7 Transaction Processing Model

The multi-process database model consists of a set $P = \{P_1, P_2, \ldots, P_n\}$ of processes executing concurrently. Processes are considered to consist of possibly several recovery blocks for rollback purposes; so that a process, in the event of a failure, will be rolled back to the beginning of the newest recovery block.

T is a set, $\{t_1, t_2, \ldots, t_m\}$ of times, such that $t_i < t_{i+1}$ for all $1 \leq i \leq m-1$, at which interactions have taken place between processes in P. Each t_i in T is associated with at least one interaction between processes in P.

The set of recovery points of any given process P_i is represented by $R_i = \{R_{ij}\}$ for $1 \leq j \leq k$ where k is the number of

recovery blocks the process P_i has entered.

An interaction tuple (t_i,P_j) associated with a process P_k represents that at time t_i processes P_k and P_j have interacted with each other. For every tuple (t_i,P_j), associated with process P_k, there is a tuple (t_i,P_k) associated with process P_j. The set of all interaction tuples associated with all the processes is a subset of the cartesian product T x P.

The progress of a process P_i is characterized by ordering, in the time domain, the union of the set of interaction tuples associated with P_i and R_i (the set of all recovery points of P_i).

For instance, from Figure 4.1, the progress of processes P_1 and P_2 can be respectively represented by the lists L_1 and L_2 given below:

$L_1 = \{R_{11},(t_2,P_2),\ (t_3,P_2),R_{12},\ (t_4,P_2),R_{13},\ (t_8,P_2),R_{14}\}.$

$L_2 = \{R_{21},\ (t_1,P_3),\ (t_2,P_1),\ R_{22},\ (t_3,P_1),\ (t_4,P_1),$
$(t_5,P_3),\ R_{23},\ (t_6,P_3),\ (t_7,P_3),\ R_{24},\ (t_8,P_1)\}.$

4.8 The Backup Algorithm

We present an algorithm to determine which specific recovery point the process which has failed is to be backed up. The scheme also determines how far other processes are to be backed out. The method assumes that the information on the interaction tuples and the progress of the processes is specified.

ALGORITHM G

Input: (i) Progress of all the processes concerned (that is, the set of all interaction tuples, and the set of all the recovery points);

(ii) the process P_k, which has failed at the current time, T_p.

Output: (i) Recovery point to which the process P_k is to be backed up; and

(ii) recovery points to which processes (if any) that may have interacted with P_k are to be backed up.

/* Data Structures */

SET: A set variable to hold the processes to be backed up. Initially it is set to the process which has failed.

BACKUP(P_i): An array of size n, which holds the recovery point to which the process P_i is to be backed up. Initially this is set to the current time, T_p. At the termination of ALGORITHM G, a process P_j with BACKUP(P_j) set to T_p needs no backing up.

PREVIOUS_BACKUP(P_i): An array of size n, which holds the previous recovery point to which the process P_i was backed up. Therefore PREVIOUS_BACKUP(P_i) is always greater than or equal to BACKUP(P_i). Initially, this array is also set to the current time, T_p.

INTERACTION_TIME(P_i): An array of size n, which holds the time at which process P_i had an interaction with some process, past its own backup point BACKUP(P_i). INTERACTION_TIME(P_i) facilitates choosing the next backup point of P_i past this interaction.

/* The Algorithm */

G1: [Initialize]

SET <-- {P_k};

for i = 1 until n do

begin

BACKUP(P_i) <-- T_p;

PREVIOUS_BACKUP(P_i) <-- T_p;

INTERACTION_TIME(P_i) <-- T_p;

end;

G2: [Select process for backing up]

if SET = ϕ then STOP

else any $P_j \in$ SET, SET <-- SET - {P_j};

and repeat steps S3, S4 for P_j.

G3: [Determine backup point]

PREVIOUS_BACKUP(P_j) <-- BACKUP(P_j);

BACKUP(P_j) <-- R_{j1}, such that

$R_{j1} \leq$ PREVIOUS_BACKUP(P_j) and

R_{j1} < INTERACTION_TIME(P_j);

G4: [Find, if any, interactions with backed up process]

For all interaction tuples associated with P_j, (t,P_i) such that BACKUP(P_j) < t < PREVIOUS_BACKUP(P_j) set SET <-- SET $\sqcup$ {P_i}; For every such P_i with (t_{i1},P_i), (t_{i2},P_i), ..., (t_{ik},P_i) where BACKUP(P_j) <t_{i1}<t_{i2}<...<t_{ik}< PREVIOUS_BACKUP(P_j), set INTERACTION_TIME(P_i) <--min{t_{i1}, INTERACTION_TIME(P_i)}.

G5: [Repeat for interacting processes (if any)]

Go To G2.

4.9 Recovery from Different Failures

The recovery aspects for the following wide spectrum of

system malfunctions is considered in this section.

(a) A storage failure (head crash);

(b) A system crash (software failure);

(c) A lost message;

(d) A duplicated message;

(e) A lost process (due to system crash and subsequent recovery);

(f) Network partitioning;

(g) Operation with missing nodes;

A head crash on disk may destroy not only the data but also the recovery data stored in the form of an audit trail or differential file. Should such a crash occur, the recreation of the recovery data is achieved by duplication of audit trails or differential files. In practice, there must be belief and dependence on some ultimate recovery data (or rather the fact that 100% reliability is not provided by any recovery data, should be recognized). However, the data damaged in a head crash can be recovered by loading a backup version and reprocessing from audit trails.

A system crash may potentially leave data mutually and internally inconsistent at the site of the failure. After the site comes up, the internal consistency of the data is maintained by either reprocessing the transactions that were active at the time of crash, or by backing out certain transactions. The principal mechanism that helps maintain mutual consistency (i.e., re-integrating the site into the system) is called "persistent communication". This

mechanism is a clean way of accomplishing re-integration of the system. Different forms of persistent communication are used [Ellis, 1977; Thomas, 1977; Hammer and Shipman, 1978]. The concept of message spoolers can be used to maintain mutual consistency with the help of messages stored when the site was down.

A lost message can be similarly obtained from the message spoolers located at alternative sites. This problem has been handled in a variety of ways in the literature. Lampson and Sturgis[1976] require minimizing the likelihood of a failure during what they call a "reliable broadcast window" (the time interval between the transmissions of the same message to different sites N_1 and N_2). Should a failure occur after the message is communicated to the site N_1, but before it is transmitted to N_2, Lampson and Sturgis propose that potential data inconsistency be detected by locks that are set (and remain set) while the site is down.

A duplicated message is easily handled, since we require processing of messages from a common source be done in their order of initiation. The uniqueness of the timestamps helps detect such duplicity.

An initiated process may be lost after recovery from a crash, due to the fact that a system failure may occcur after the updates are committed but before they are recorded for recovery. This is handled by requiring that the recovery data be recorded using write ahead log protocol, as

explained earlier.

The problem is more serious in the event of brief communication failures which result in isolated subclusters of computers. Each separated segment continues its operation, without any chance for the isolated pieces to coordinate their activities. Consequently, on restoration of communication the fragments are inconsistent. Transactions run in a partitioned network are simply incorrect in the context of the total network. Thus, interleaving their actions to maintain consistency is a futile effort. The correct action depends on the database semantics, topology of the partition, and the actions of the transactions. A complete technical solution without human decision making does not seem feasible.

System operation with missing sites (failed sites) is necessary to avoid delaying transactions till recovery occurs. This can be accomplished by recording update messages for maintaining mutual consistency in message spoolers, for the failed sites. Lampson and Sturgis use the concept of "intentions list" (a non-volatile storage where all updates are recorded), and the fact that set locks leave a lingering recollection of a failed site until it recovers.

4.10 Discussion

Not every issue involved in system recovery is discussed rigorously in this chapter. In particular,

transactions have been subdivided into classes depending on their recovery requirements. Appropriate system recovery protocols for transaction classes have been proposed. We have stressed the importance of the availability of the on-line system at all times, and advocate that the recovery overhead should be directly dependent on the value or sensitivity of the data accessed. To accomplish this we have used our knowledge of the nature of all the transactions that access the data. Presumably the knowledge available about the concurrent processes can be used in designing optimal algorithms for several other issues involved in the design of distributed databases. It is necessary to conduct experiments and subsequent analysis of simulated or distributed databases, to provide better assessment of the techniques. Performance measurements should be the next big step in distributed database research.

CHAPTER 5

CONCLUDING REMARKS

5.1 Summary of Results Obtained

5.1.1 Deadlocks

The desire for complete understanding of the deadlock problem in the context of operating systems, databases and distributed databases is motivated. In particular, the interrelationships and characteristics of the problem in three broad fields are brought out with the aid of a series of good examples. It is argued that deadlock detection schemes are better suited than avoidance or prevention mechanisms for distributed systems. The importance of the combined approach incorporating detection, avoidance, and prevention principles is demonstrated.

A new approach to deadlock detection in a network environment is proposed. The concept of "on-line" deadlock detection in distributed information networks is introduced. It is defined to be the process of recognizing deadlock occurrence as soon as it happens, at the installation which makes the resource allocation decision, without the necessity of further communication for every request made or

granted. An on-line detection algorithm is suggested and developed. All of the earlier algorithms restrict a process to having at most one outstanding request. In our approach, such a restriction is removed in view of the fact that in real-world applications more than one outstanding request is a certainty. This leads to a situation in which an allocation decision on a data resource (with multiple waiting-access requests) released by a completing process would lead to a deadlock. For this case, an elegant solution which combines the principles of detection and avoidance is shown where a potential deadlock is detected and avoided. On account of the fact that requests in databases are data-driven and content-based, the possibility of multiple outstanding requests is high. Thus, the results and the mixed solution in this case are new and original. Besides its low level of communication activity the approach has several other major advantages as outlined in Section 3.9.

5.1.2 System Recovery

Another aspect of the problem considered is the reliable operation of database systems, partitioned and/or replicated over a network of computers. The design of a method which maintains database consistency during system update and recovery is guided by the goals of simplicity, tolerable overhead, partial operability, and avoidance of global rollback. In this new approach, retrieval and update transactions are subclassified and recovery protocols

defined which take advantage of the known properties of each transaction class. An optimal policy for checkpointing in a particular recovery protocol is derived using a simple model. The policy determines the checkpoint dynamically as the transactions are processed, and is different from earlier fixed interval approaches. All the parameters involved can be pre-determined due to the transaction classification facility, database usage statistics and the expected behaviour of certain transactions. Consequently, the approach is new and practical. The cascading effect of a global rollback is modeled by using the progress of processes represented by a set of interaction tuples and recovery points ordered in the time domain. A backup algorithm based on this model is developed. Recovery aspects for a wide set of system failures are considered and several partial solutions are outlined.

5.2 Significance and Motivation

With the growing use of terminal oriented computer systems, and the increasing trend by commercial firms for real-time operations, especially those involving databases, the database-oriented operating systems are faced with heavy demands. A characteristic of such contemporary systems is their high degree of resource and data sharing. Consequently, the possibility of deadlocks increases. Also, the problem of system recovery from crashes is intensified.

In our view, deadlock prevention schemes are not

justifiable for use in distributed databases. Extensive coordination and substantial communication is necessary before process initiation, for processes that cannot be shown to be nonconflicting. This affects system performance by lowering the degree of concurrency. The past use of prevention principles was acceptable because of low levels of concurrency in systems rather than any inherent superiority.

We advocate the use of deadlock detection in distributed systems [Isloor and Marsland, 1978; Marsland and Isloor, 1978]. Our views are also supported in recent literature. When dealing with concurrent database accesses, little is known about the probability of interference or deadlock. For transaction processing systems, it is firmly believed that interference is rare and that elaborate avoidance or prevention mechanisms would not be economical [Peebles and Manning, 1978]. Further, to quote [Le Lann, 1978] again, "our conclusion will be that for systems which include a partitioned database and which provide for storage of pending requests, maintenance of internal integrity boils down to a problem of deadlock avoidance or detection with distributed control". As a consequence, and in view of the present day trend towards increased concurrent access in systems, the on-line detection technique is a step toward increasing user confidence in distributed systems. The existing algorithms for deadlock detection in distributed databases cannot be used for on-line detection because:

(i) for every request granted or not, these algorithms need to obtain the global network status by simultaneous communication of the status of each installation, which leads to a tremendous amount of communication in the case of on-line detection;

(ii) such huge communication traffic results in synchronization problems due to communication delays in which either a deadlock is indicated where one no longer exists, or an existing deadlock goes undetected; and

(iii) after obtaining the complete network status, the algorithms have to perform computations to detect a deadlock.

In a distributed environment, a longer delay in the detection of deadlocks can have disastrous effects on the consistency of the database. By detecting on-line, closer to the source and instant of occurrence, the opportunity for timely corrective action is greatly enhanced.

The basic strategy expounded in the design of recovery protocols consists of discriminating between update and retrieval transactions, and sub-classifying them in such a way that recovery protocols can be developed for each type of transaction. It is believed that for transaction processing systems over 95-99% of transactions fall into the

category of local interactions only[1]. Predefinition of transaction classes would normally be done by the database administrator, based on either information about the special application, or by gathering usage statistics about the actual value of the occurring transactions. However, a new category is created for transactions for which insufficient classification data exists, or unpredictable or unexpected behaviour is possible. Thus, our strategy used for recovery is on the similar lines of thought as the current day trends.

5.3 Directions for Research

It is difficult to estimate the performance effects of deadlock handling techniques or the probability of occurrence of deadlocks in distributed databases, since communication time is a critical factor. Because of the increased complexity of distributed databases a significant factor in handling deadlocks is the operational efficiency. It is probably necessary for distributed databases to become a commercial reality, so that experimental data can be gathered to measure performances and probabilities, before

---

1: See

P.A. Bernstein et al., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)", IEEE Trans. on Software Engg., SE-4 (3), May 1978, pp. 154-168.

M.R. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", IEEE Trans. on Software Engg., SE-5(3), May 1979, pp. 188-194.

the effect of communication aspects can be estimated. To estimate the performance of the on-line detection scheme, future researchers should look for appropriate simulation, testing, and interpretation.

The probabilistic model of deadlocks [Ellis, 1974] considered no more than 5 resources and processes, which is trivial in the commercial world. More research is clearly needed in that area. A comprehensive probabilistic model for computer deadlocks of large systems is missing from the literature. More research is warranted to include extensions of any such comprehensive models to systems with consumable resources.

A comprehensive combined approach to deadlocks in database systems or distributed databases is probably a welcome step. Deadlock prevention techniques are advisable for processes accessing highly secure data in a concurrent system, whereas detection and avoidance principles can be used for less important processes. The literature has not yet included solutions to deadlock problems involving processes accessing highly classified data in an integrated database. Research is necessary to determine an efficient and effective method of rolling back a process. Such a mechanism can make deadlock detection techniques much more attractive.

As more and more data is integrated over a network of computers, resulting in the databases becoming more

accessible to larger number of diverse application jobs, the complexity of the function of the database administrator (DBA) increases enormously. The actions of the operating system (which manages application jobs) and that of the DBA (who maintains process integrity and the consistency of the database), have to be coordinated. It is necessary that both the DBA and the operating system have a thorough understanding of the relationships among concurrency controls, processors, processes, deadlock handling and recovery techniques, communication aspects and protocols. The relevance of such coordination may necessitate having a higher authority over both the DBA and the operating systems. This area calls for deeper study especially in a network environment.

Formal development and analysis of the recovery protocols proposed is necessary. Such analysis should presumably use the knowledge available about the concurrent processes. Performance measurements following the analysis, on simulated or implemented systems should be the next big step in distributed database research.

The detection of errors and failures, and their categorization offer open areas for research. The opportunity for timely correction of errors is greatly enhanced if they are detected closer to the source of their occurrence. Propagation of errors is a major problem and is severe in a network environment. This calls for immediate detection at the source to maintain a high degree of

performance. The extents of applicability of hardware and software recoverability techniques so as to speed up system recovery needs further research.

Data reliability in systems with highly sensitive data needs multiple levels of protection and recoverability. This ensures that an error getting past one level of detection does not corrupt the database and contribute to the propagation of corrupt data. An elaborate feedback scheme to provide such protection would involve presenting parts of the sensitive database for human scrutiny. The extent and manner of human participation with "intelligent" detection systems for data reliability warrants further research.

BIBLIOGRAPHY

Adiba, M., et al., "Issues in Distributed Data Base Management Systems: A Technical Overview", Proc. Fourth International Conf. on Very Large Data Bases, Berlin, West Germany, October 1978, pp. 89-110.

Astrahan, M.M., et al., "System R: Relational Approach to Database Management", ACM Trans. on Database Systems, 1(2), June 1976, pp. 97-137.

Banerjee, J., Baum, R.I., and Hsiao, D.K., "Concepts and Capabilities of a Database Computer", ACM Trans. on Database Systems, 3(4), December 1978, pp. 347-384.

Baum, R.I., and Hsiao, D.K., "Database Computers- A Step Towards Data Utilities", IEEE Trans. on Computers, C-25(12), December 1976, pp. 1254-1259.

Bayer, R., "On the Integrity of Data Bases and Resource Locking", in Lecture Notes in Computer Science 39, Proc. 5th Informatik Symposium, Germany, H. Hasselmeier, W.G. Spruth,(eds.), September 1975, pp. 339-361.

Bayer, R., "Integrity, Concurrency, and Recovery in Databases", in Lecture Notes in Computer Science 44, Proc. ECI Conf., Germany, K. Samelson,(ed.), 1976, pp. 79-105.

Bernstein, P.A., et al., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case).", IEEE Trans. on Software Engineering, SE-4 (3), May 1978, pp. 154-168.

Booth, G.M., "The Use of Distributed Data Bases in Information Networks", Proc. First International Conf. on Computer Communication: Impacts and Implications, October 1972, pp. 371-376.

Booth, G.M., "Distributed Information Systems", Proc. AFIPS National Computer Conf., 45, June 1976, pp. 789-794.

Canaday, R.H., et al., "A Back-end Computer for Data Management", CACM, 17(10), October 1974, pp. 575-582.

Chamberlin, D.D., "Relational Data-Base Management Systems", Computing Surveys, 8(1), March 1976, pp. 43-66.

Chamberlin, D.D., Boyce, R.F., and Traiger, I.L., "A Deadlock-free Scheme for Resource Locking in a Data-Base

Environment", Information Processing 74, Proc. IFIP Congress, North-Holland Publishing Co., Amsterdam, August 1974, pp. 340-343.

Chandra, A.N., Howe, W.G., and Karp, D.P., "Communication Protocol for Deadlock Detection in Computer Networks", IBM Technical Disclosure Bulletin, 16(10), March 1974, pp. 3471-3481.

Chandy, K.M., and Ramamoorthy, C.V., "Rollback and Recovery Strategies", IEEE Trans. on Computers, C-21(2), February 1972, pp. 137-146.

Chandy, K.M., Browne, J.C., Dissly, C.W., and Uhrig, W.R., "Analytic Models for Rollback and Recovery in Data Base Systems", IEEE Trans. on Software Engineering, SE-1(1), March 1975, pp. 100-110.

Chang, S.K., "A Model for Distributed Computer System Design", IEEE Trans. on Systems, Man, and Cybernetics, SMC-5 (6), May 1976, pp. 344-359.

Chu, W.W., and Ohlmacher, G., "Avoiding Deadlock in Distributed Data Bases", Proc. ACM National Conf., 1, November 1974, pp. 156-160.

Clipsham, W.W., Glave, F.E., and Narraway, M.L., "Datapac Network Overview", Proc. Third International Conf. on Computer Communications, Toronto, Canada, August 1976, pp. 131-136.

The CODASYL Systems Committee, "Distributed Data Base Technology - An Interim Report of the CODASYL Systems Committee", Proc. AFIPS National Computer Conf., 47, June 1978, pp. 909-917.

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", CACM, 13(6), June 1970, pp. 377-387.

Codd, E.F., "Normalized Data Base Structure: A Brief Tutorial", Proc. 1971 ACM-SIGFIDET Workshop on Data Definition, Access, and Control, November 1971, pp. 1-17.

Codd, E.F., "Relational Completeness of Data-Base Sublanguages", Courant Computer Science Symposia 6, Data Base Systems, Prentice Hall, Englewood Cliffs, N.J., 1971, pp. 65-98.

Coffman, Jr., E.G., Elphick, M.J., and Shoshani, A., "System Deadlocks", Computing Surveys, 3(2), June 1971, pp. 67-78.

Comba, P.G., "Needed: Distributed Control", Proc. International Conf. on Very Large Data Bases, 1(1), Framingham, Mass., September 1975, pp. 364-375.

Davenport, R.A., "Distributed Database Technology - A Survey", Computer Networks, 2(3), July 1978, pp. 155-167.

Deppe, M.E., and Fry, J.P., "Distributed Data Bases: A Summary of Research", Computer Networks, 1(2), September 1976, pp. 130-138.

Devillers, R., "Game Interpretation of the Deadlock Avoidance Problem", CACM, 20(10), October 1977, pp. 741-745.

Eckhouse, R.H., Stankovic, Jr., J.A., and van Dam, A., "Issues in Distributed Processing - An Overview of Two Workshops", IEEE Computer, 11(1), January 1978, pp. 22-26.

Ellis, C.A., "Probabilistic Models of Computer Deadlock", Report # CU-CS-041-74, Dept. Of Computer Science, University of Colorado, Boulder, U.S.A., April 1974. (25 pages)

Ellis, C.A., "A Robust Algorithm for Updating Duplicate Databases", Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, U.S.A., May 1977, pp. 146-158.

Enslow, Jr., P.H., "What is a 'Distributed' Data Processing System?", IEEE Computer, 11(1), January 1978, pp. 13-21.

Eswaran, K.P., Gray, J.N., Lorie, R.A., and Traiger, I.L., "The Notions of Consistency and Predicate Locks in a Data Base System", CACM, 19(11), November 1976, pp. 624-633.

Goldman, B., "Deadlock Detection in Computer Networks", Technical Report MIT/LCS/TR-185, Laboratory for Computer Science, M.I.T., Cambridge, Mass., September 1977. (180 pages)

Gray, J.N., "Locking in a Decentralized Computer System", IBM Research Report RJ 1346, IBM Research Laboratory, San Jose, Calif., February 1974. (59 pages)

Gray, J.N., Lorie, R.A., and Putzolu, G.R., "Granularity of Locks in a Shared Data Base", Proc. International Conf. on Very Large Data Bases, Framingham, Mass., September 1975, pp. 428-451.

Gray, J.N., Lorie, R.A., Putzolu, G.R., and Traiger, I.L., "Granularity of Locks and Degrees of Consistency in a Shared Data Base", in Modeling in Data Base Management Systems, G.M. Nijssen, (ed.), North-Holland Publishing Co., Amsterdam, 1976, pp. 365-394.

Gray, J.N., "Notes on Data Base Operating Systems", in Operating Systems (An Advanced Course) in Lecture Notes in Computer Science 60, R. Bayer et al., (eds.), 1978, pp. 393-481.

Habermann, A.N., "Prevention of System Deadlocks", CACM, 12 (7), July 1969, pp. 373-377,385.

Hammer, M., and Shipman, D., "An Overview of Reliability Mechanisms for a Distributed Data Base System", Digest of Papers from Spring COMPCON 78, San Francisco, Calif., U.S.A., February-March 1978, pp. 63-65.

Havender, J.W., "Avoiding Deadlock in Multitasking Systems", IBM Systems Journal, 7(2), 1968, pp. 74-84.

Hebalkar, P.G., "Deadlock-free Resource Sharing in Asynchronous Systems", Ph.D. Dissertation, Electrical Engineering Dept., M.I.T., Cambridge, Mass., September 1970.

Holt, R.C., "On Deadlock in Computer Systems", Ph.D. Dissertation, Dept. of Computer Science, Cornell University, Ithaca, N.Y., January 1971. (Also as Technical Report CSRG-6, Computer Systems Research Group, University of Toronto, Toronto, Canada, April 1971.)

Holt, R.C., "Some Deadlock Properties of Computer Systems", Computing Surveys, 4(3), September 1972, pp. 179-196.

Hovey, R.B., "The User's Role in Connecting to Value Added Networks", Data Communications, May/June 1974.

Howard, Jr., J.H., "Mixed Solutions for the Deadlock Problem", CACM, 16(7), July 1973, pp. 427-430.

Hsiao, D.K., (Guest Editor), "Data Base Machines", (Special Issue), IEEE Computer, 12(3), March 1979, pp. 7-79.

Isloor, S.S., and Marsland, T.A., "Deadlock Detection in Databases Distributed on a Network of Computers", Technical Report TR 78-3, Dept. of Computing Science, The Univ. of Alberta, Edmonton, Canada, May 1978. (40 pages)

King, P.F., and Collmeyer, A.J., "Database Sharing: An Efficient Mechanism for Supporting Concurrent Processes", Proc. AFIPS National Computer Conf., 42, June 1973, pp. 271-275.

Lampson, B., and Sturgis, H., "Crash Recovery in a Distributed Data Storage System", Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto, California, U.S.A., 1976. (also to appear in CACM).

Le Lann, G., "Distributed Systems-Towards a Formal Approach", Information Processing 77, Proc. IFIP Congress, North-Holland Publishing Co., Amsterdam, August 1977, pp. 155-160.

Le Lann, G., "Pseudo-Dynamic Resource Allocation in Distributed Databases", Proc. Fourth International Conf.

on Computer Communications, ICCC-78, Kyoto, Japan, September 1978, pp. 245-251.

Lientz, B.P., and Weiss, I.R., "Trade-offs of Secure Processing in Centralized versus Distributed Networks", Computer Networks, 2(1), February 1978, pp. 35-43.

Lomet, D.B., "A Practical Deadlock Avoidance Algorithm for Data Base Systems", Proc. ACM-SIGMOD International Conf. on Management of Data, Toronto, Canada, August 1977, pp. 122-127.

Lowenthal, E.I., "A Survey - the Application of Data Base Management Computers in Distributed Systems", Proc. Third International Conf. on Very Large Data Bases, Tokyo, Japan, October 1977, pp. 85-92.

Mahmoud, S.A., and Riordon, J.S., "Protocol Considerations for Software Controlled Access Methods in Distributed Data Bases", Proc. International Symposium on Computer Performance Modeling, Measurement and Evaluation, Cambridge, Mass., March 29-31, 1976, pp. 241-264.

Mahmoud, S.A., and Riordon, J.S., "Software Controlled Access to Distributed Data Bases", INFOR, 15(1), February 1977, pp. 22-36.

Marsland, T.A., and Isloor, S.S., "A Review of the Deadlock Problem in Operating, Database, and Distributed Systems", Technical Report TR 78-5, Dept. of Computing Science, The Univ. of Alberta, Edmonton, Canada, December 1978. (62 pages)

Marsland, T.A., and Sutphen, S.F., "Design and Experience with a Distributed Computing System", Proc. Canadian Computer Conf., CIPS Session' 78, Edmonton, Canada, May 1978, pp. 367-371.

Maryanski, F.J., "A Deadlock Prevention Algorithm for Distributed Data Base Management Systems", Technical Report CS 77-02, Computer Science Dept., Kansas State University, Manhattan, Kansas, February 1977. (24 pages)

Maryanski, F.J., "A Survey of Developments in Distributed Data Base Management Systems", IEEE Computer, 11(2), February, 1978, pp. 28-38.

Maryanski, F.J., and Fisher, P.S., "Rollback and Recovery in Distributed Data Base Management Systems", Technical Report CS 77-05, Computer Science Dept., Kansas State University, Manhattan, Kansas, February 1977. (19 pages)

Maryanski, F.J., and Kreimer, D.E., "Effects of Distributed Processing in a Data Processing Environment", Computer Science Dept., Kansas State University, Manhattan, Kansas,

1978.

McQuillan, J.M., and Walden, D.C., "The ARPA Network Design Decisions," Computer Networks, 1(5), August 1977, pp. 243-289.

Metcalfe, R.M., and Boggs, D.R., "Ethernet: Distributed Packet Switching for Local Computer Networks", CACM, 19(7), July 1976, pp. 395-404.

Peebles, R., and Manning, E., "System Architecture for Distributed Data Management", IEEE Computer, 11(1), January 1978, pp. 40-47.

Randell, B., "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engg., SE-1(2), June 1975, pp. 220-232.

Rosenkrantz, D.J., Stearns, R.E., and Lewis, P.M. II, "System Level Concurrency Control for Distributed Database Systems", ACM Trans. on Database Systems, 3(2), June 1978, pp. 178-198.

Rosenkrantz, D.J., "Dynamic Database Dumping", Proc. 1978 ACM-SIGMOD International Conf. on Management of Data, Austin, Texas, U.S.A., June 1978, pp. 3-8.

Rothnie, J.B., and Goodman, N., "A Survey of Research and Development in Distributed Data Base Management", Proc. Third International Conf. on Very Large Data Bases, Tokyo, Japan, October 1977, pp. 48-62.

Rothnie, J.B., Goodman, N., and Marill, T., "Database Management in Distributed Networks", (Chapter 10), Protocols and Techniques for Data Communications Networks, F.F. Kuo, (ed.), Prentice Hall, Englewood Cliffs, N.J., 1979 (in Press).

Saltzer, J.H., and Schroeder, M.D., "The Protection of Information in Computer Systems", Proceedings of the IEEE, 63(9), September 1975, pp. 1278-1308.

Schlageter, G., "Access Synchronization and Deadlock Analysis in Database Systems: An Implementation-Oriented Approach", Information Systems, 1(2), 1975, pp. 97-102.

Severance, D.G., and Lohman, G.M., "Differential Files: Their Application to the Maintenance of Large Databases", ACM Trans. on Database Sysems, 1(3), September 1976, pp. 256-267.

Shoshani, A., and Bernstein, A.J., "Synchronization in a Parallel Accessed Data Base", CACM, 12(11), November 1969, pp. 605-607.

Stearns, R.E., Lewis, P.M. II, and Rosenkrantz, D.J., "Concurrency Control for Database Systems", Proc. IEEE 17 th Annual Symposium on Foundations of Computer Science, October 1976, pp. 19-32.

Stonebraker, M.R., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", Memorandum No. UCM/ERL M78/24, Electronics Research Lab., Univ. of California, Berkeley, Calif., U.S.A., May 1978. (24 pages)

Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Data Baes", Report No. 3733, Bolt Beranek and Newman Inc., Cambridge, Mass., U.S.A., December 1977. (61 pages)

van Dam, A., Stankovic, Jr., J., (Guest Editors), "Distributed Processing", (Special Issue), IEEE Computer, 11 (1), January 1978, pp. 10-57.

Verhofstad, J.S.M., "Recovery and Crash Resistance in a Filing System", Proc. 1977 ACM-SIGMOD International Conf. on Management of Data, Toronto, Canada, August 1977, pp. 158-167.

Verhofstad, J.S.M., "Recovery Techniques for Database Systems", Computing Surveys, 10(2), June 1978, pp. 167-195.

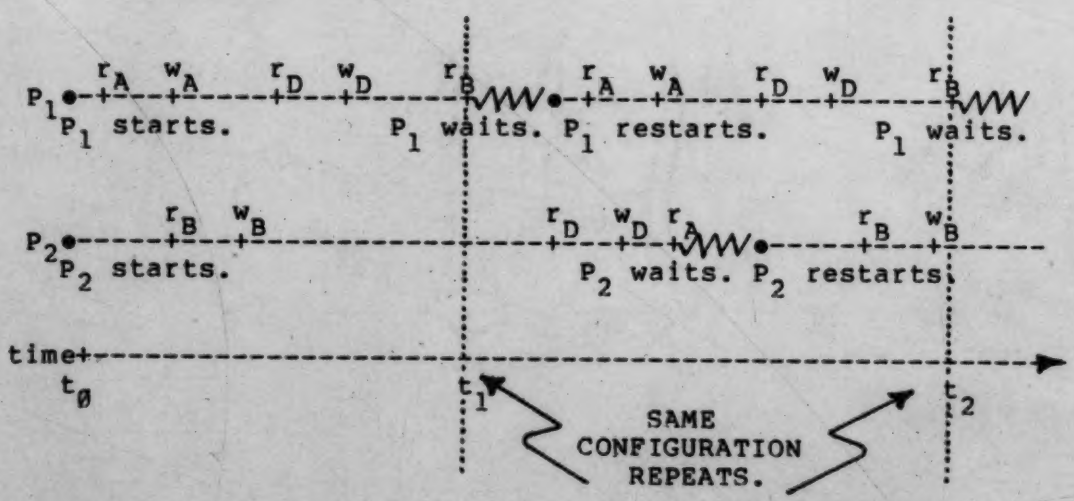
Warshall, S., "A Theorem on Boolean Matrices", JACM, 9(1), January 1962, pp. 11-12.

Zeigler, K., Jr., "Distributed Data Base, Where are You?-(A Tutorial)", Information Processing 77, Proc. IFIP Congress, North-Holland Publishing Co., Amsterdam, August 1977, pp. 113-115.

APPENDIX A

AN EXAMPLE OF "CYCLIC RESTART"

Certain prevention methods require a blocked process to release a resource held, when requested by an active process. Also an active process gets blocked when it requests a resource held by another active process. For some peculiar situations in database systems, this scheme is subject to "cyclic restart" in which two or more processes get entangled perpetually by continually blocking, aborting, and restarting each other. We demonstrate this phenomenon with an example.



Consider two processes P_1 and P_2 starting at time $t_\emptyset$. Processes P_1 and P_2 read and write entities {A,D} and B respectively. At time t_1 ($t_\emptyset < t_1$) process P_1 requests access to entity B and gets blocked, whereas shortly afterwards P_2

requests access to D resulting in the abortion and restarting of P_1 and so on. The situation at the time t_2 is identical to that at the time t_1. This cycle will try to self-sychronize itself and will tend to repeat this loop indefinitely notwithstanding minor system environment variations in timing.

APPENDIX B

AN EXAMPLE FOR MULTIPLE OUTSTANDING REQUESTS

To show the possibility of more than one outstanding request per process, we choose "Presidential Data Base" [Chamberlin, 1976] in relational model of data [Codd, 1970]. The relations in the database are in Third Normal Form [Codd, 1971a]. A query to the database, which causes two simultaneous requests, is expressed in Data Sub-Language ALPHA [Codd, 1971b].

The relations in the database and the query are:

R_1: ELECTIONS_WON (YEAR, WINNER_NAME, WINNER_VOTES)

R_2: PRESIDENTS (NAME, PARTY, HOME_STATE)

R_3: ELECTIONS_LOST (YEAR, LOSER_NAME, LOSER_VOTES)

R_4: LOSERS (NAME, PARTY)

Query: (a) (In English) List the election years in which a Republican from Illinois was elected.

The intent of this query in English is: Retrieve the YEAR attribute from any tuple of relation ELECTIONS_WON whose WINNER_NAME attribute matches the NAME attribute of a tuple of relation PRESIDENTS if that tuple of relation PRESIDENTS has a HOME_STATE of ILLINOIS and a PARTY equalling REPUBLICAN.

(b) (In DSL ALPHA)

```
RANGE PRESIDENTS P
RANGE ELECTIONS_WON E
```

GET W E.YEAR: $\exists$ P (P.NAME = E.WINNER_NAME AND
P.PARTY = 'REPUBLICAN' AND
P.HOME_STATE = 'ILLINOIS')

For the execution of this query it is essential to gain access to the relations R_2 and R_1 at the same time, causing two potential outstanding requests.